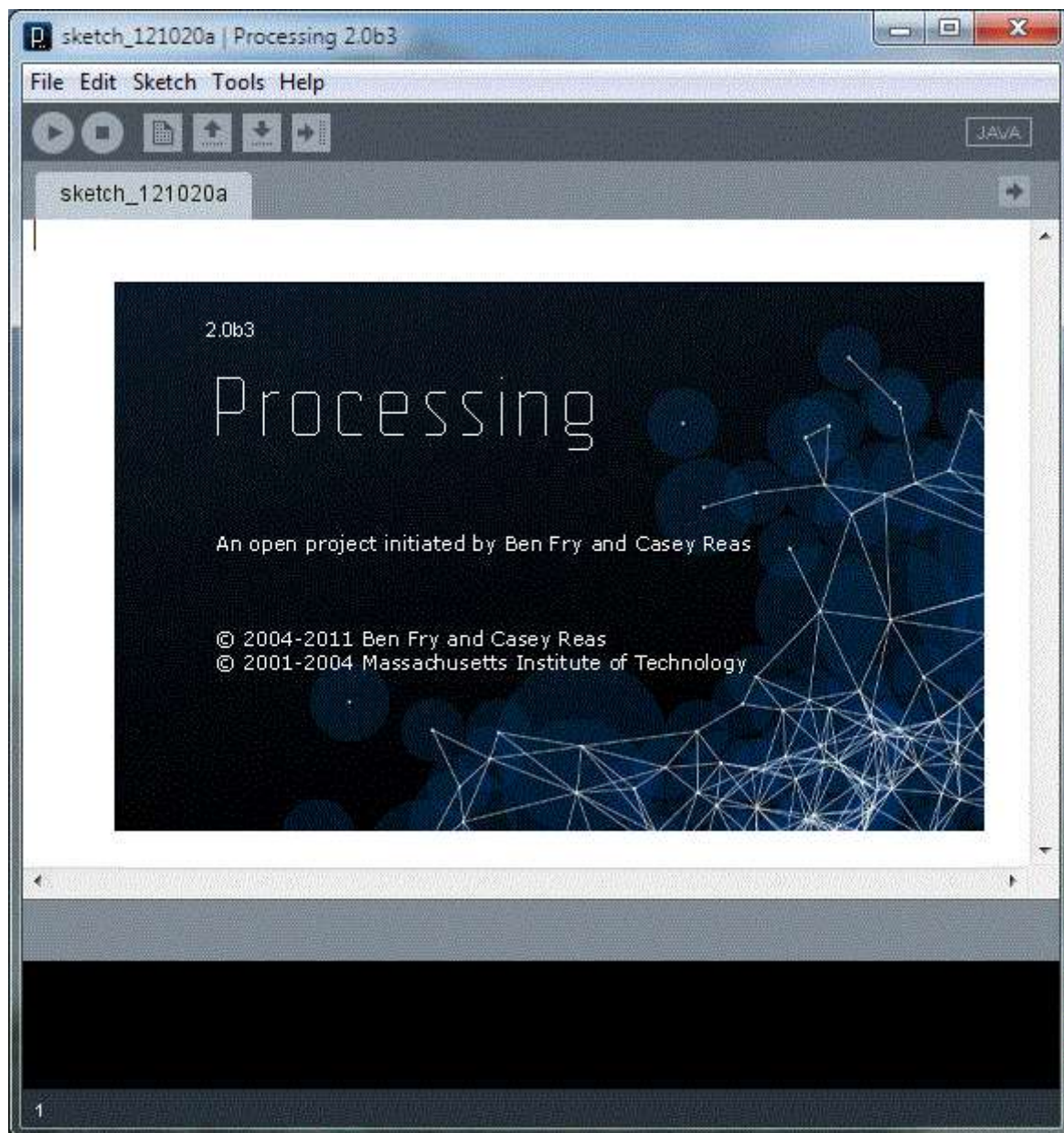


PASQUALE PIETRANGELO



PROCESSING

WWW.UBERTINI.IT

MODULO 1

PROCESSING

Linguaggio
Struttura del programma
Strutture di controllo
Funzioni
Array
OOP
File
Grafica
GUI
3D
I/O
Robot
Musica
Video
HTML 5.0
Network

LINGUAGGIO

INTRODUZIONE

Processing è un linguaggio di programmazione specifico per programmi di grafica, realizzazione di animazioni, immagini e interazioni.

È stato sviluppato come progetto Open Source da due ricercatori del **MIT** (*Massachusetts Institute of Technology*) Media Lab, Ben Fry e Casey Reas, come strumento per insegnare le basi della programmazione in un ambiente visuale e d'impatto immediato.

È un prodotto Open Source disponibile per le piattaforme Windows, Macintosh e Linux, scaricabile dal sito ufficiale.

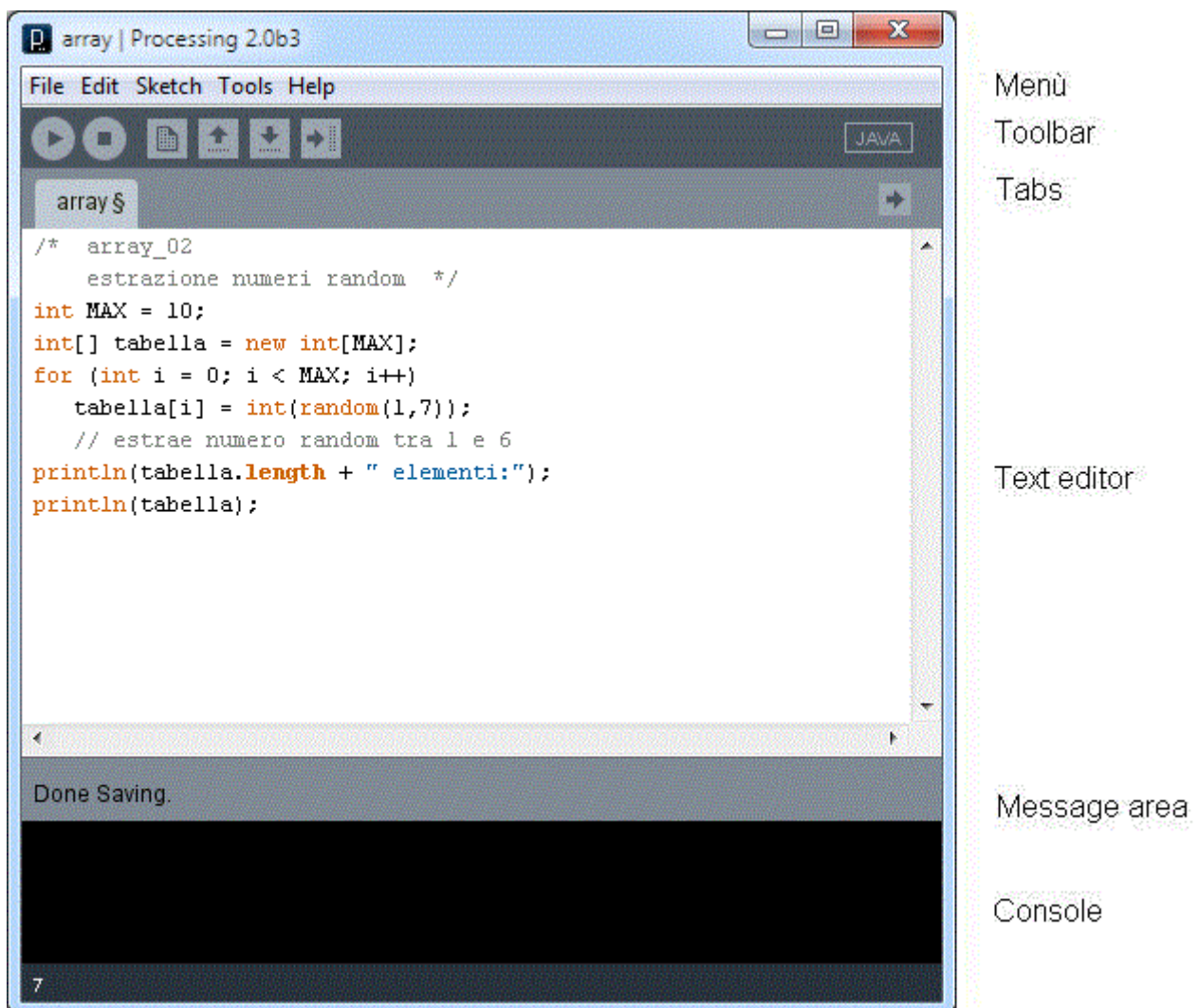
Si basa sulla tecnologia Java: uno sketch o programma Processing è tradotto in un programma Java che è a sua volta compilato e infine eseguito.

Tutto ciò avviene in modo trasparente all'utente, quando si preme il tasto **Run**.

Insieme al linguaggio è distribuito un ambiente **IDE** (*Integrated Development Environment*).

Sono forniti esempi visualizzabili dal menu **File/Examples**.

IDE



Prima di entrare nei dettagli del linguaggio analizziamo il **PDE** (*Processing Development*

Environment).

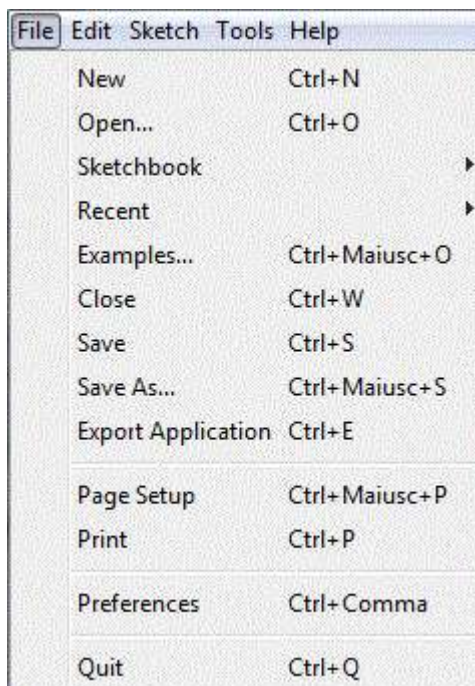
In alto abbiamo i comandi da menu, toolbar, tabs.

Al centro abbiamo l'editor di testo.

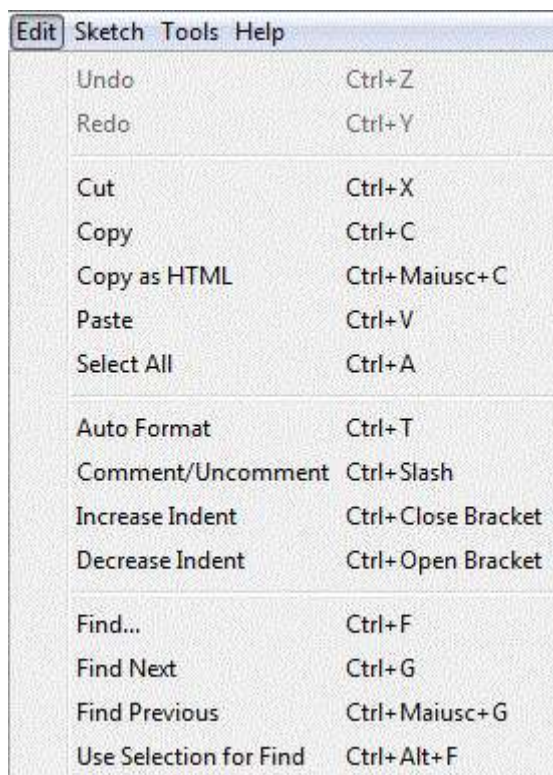
In basso la finestra di console, sulla quale sono visualizzati eventuali errori di compilazione e i messaggi di errore che avvengono a run-time.

COMANDI DA MENU

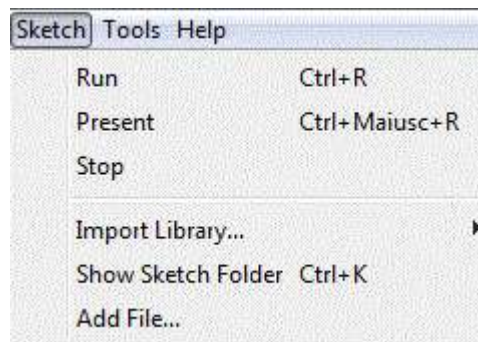
File: comandi per gestire ed esportare file.



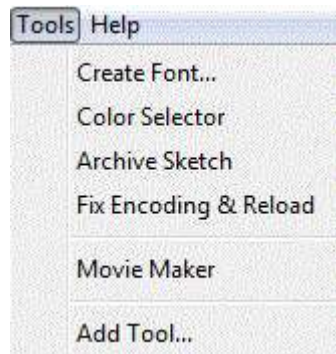
Edit: controlli per l'editor.



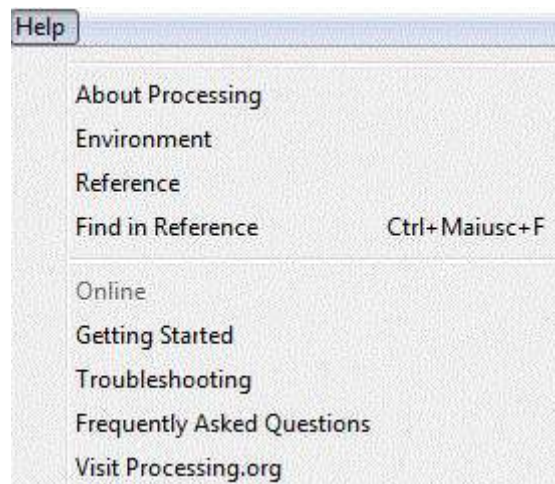
Sketch: esecuzione e stop dei programmi.



Tools: assistenza nell'uso di Processing.



Help: reference file per linguaggio e l'ambiente.



Toolbar

Run: compila, apre display window e avvia il programma.

Stop: termina il programma ma non chiude la display window.

New: crea un nuovo sketch.

Open: apre lo sketch.

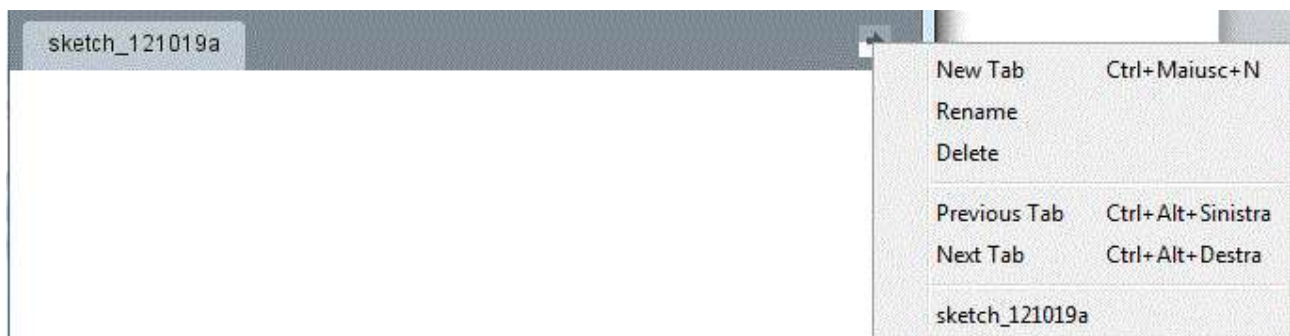
Save: salva lo sketch corrente nella sua locazione.

Export: permette di esportare lo sketch come Applet Java.

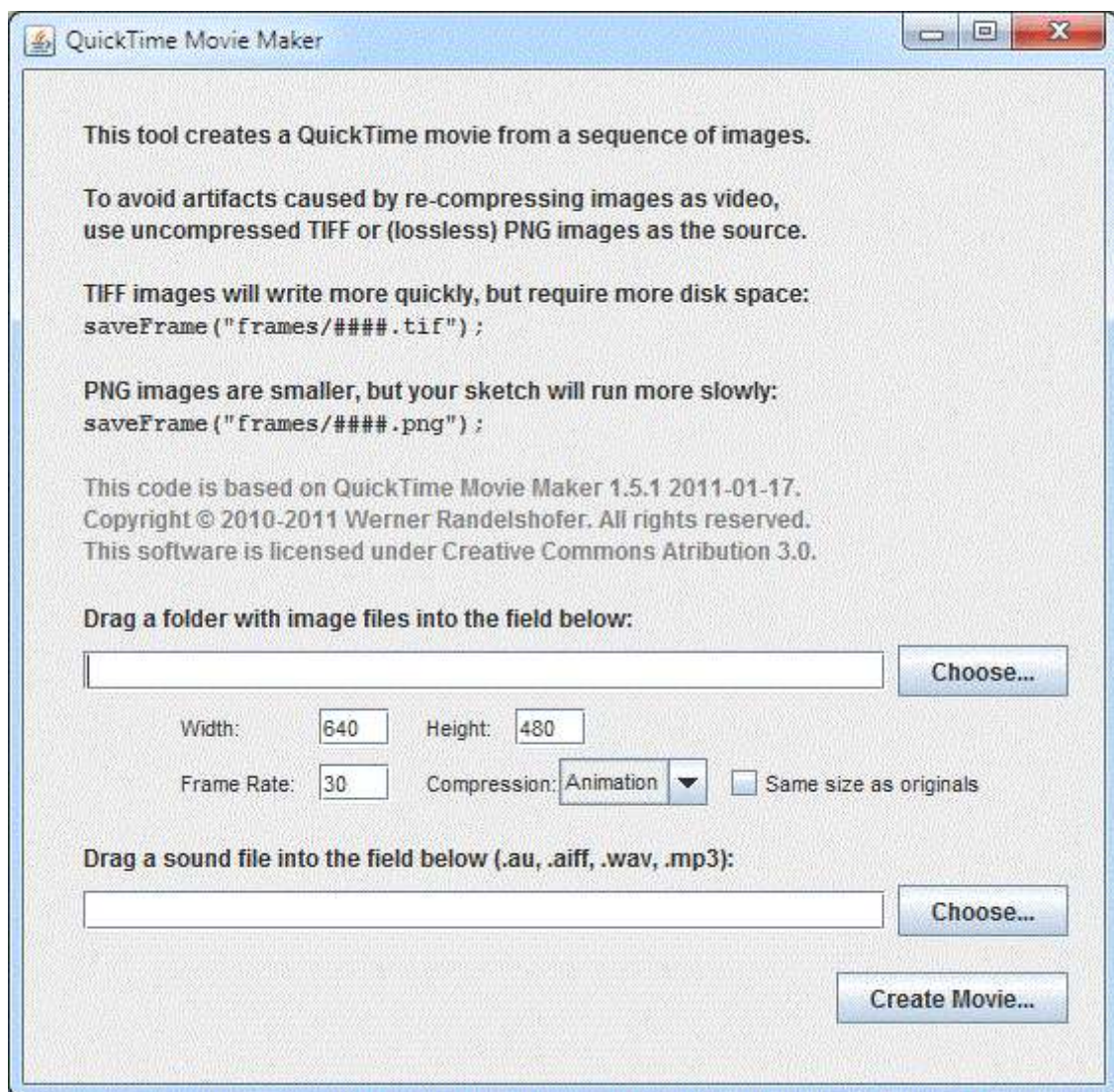


Tabs

Crea una nuova scheda, rinomina o cancella la scheda corrente.



Fare clic su **Tools/Movie Maker**.



MODALITÀ

In Processing è possibile adottare due modalità di programmazione.

1. **Basic**: tutte le operazioni sono eseguite in sequenza, è eventualmente disegnato qualcosa sullo schermo e poi il codice non è più eseguito; non è possibile quindi fare animazioni o interagire con l'utente, ma solo fare disegni statici.
2. **Continuous**: è possibile realizzare animazioni; per farlo è necessario scrivere due metodi.
 - 2.1. **setup** che conterrà tutte le operazioni che devono essere eseguite una sola volta

all'inizio del programma.

- 2.2. ***draw*** è il ciclo principale del programma; è eseguito in continuazione e contiene tutti i comandi per disegnare le animazioni; di norma il metodo *draw* è eseguito 60 volte il secondo o in numero minore se l'esecuzione dovesse essere più lenta, maggiore se l'esecuzione dovesse essere più veloce; è possibile cambiare la velocità con il comando *frameRate*.

Tale modalità può essere alterata attraverso i metodi *loop*, *noloop* e *redraw*.

STRUTTURA DEL PROGRAMMA

INTRODUZIONE

Un programma scritto in Processing è detto sketch; il file principale di uno sketch ha lo stesso nome della cartella.

Esempio.

SKETCH SKETCH_123.

Cartella SKETCH_123.

File principale nella cartella: SKETCH_123.PDE.

Ogni programma ha la seguente struttura.

✓ Dichiarazione delle variabili globali del programma.

✓ Dichiarazione dei metodi del programma.

I metodi possono essere classificati in tre categorie.

1. *setup* e *draw* sono due esempi di metodi che vanno solo dichiarati perché sono chiamati in modo implicito da Processing e costituiscono un caso di metodi detti **callback**, chiamati dal sistema in modo autonomo o quando si verificano particolari situazioni; il metodo *setup* contiene le istruzioni d'inizializzazione, in altre parole di preparazione al disegno come il dimensionamento della finestra, l'impostazione dei colori di sfondo e di disegno; il metodo *draw* contiene le istruzioni di disegno nella finestra del programma; se il metodo *noLoop* è invocato all'interno del metodo *setup* allora il metodo *draw* è eseguito una sola volta, altrimenti è eseguito ripetutamente, all'infinito, fino a che il programma non è terminato con il pulsante **Stop** oppure eseguendo il metodo *exit*; un caso importante di metodi callback sono i metodi ascoltatori di eventi che sono eseguiti quando si verifica uno specifico evento, ad esempio il clic del mouse
2. Altri metodi sono predefiniti, ad esempio i metodi *stroke* e *line* ma vanno solo chiamati, passando loro i parametri opportuni.
3. I metodi definiti dall'utente sono quelli, come il metodo *disegnaLinea* che l'utente definisce e invoca.

ESEMPIO

S'itera l'esecuzione di un metodo *disegnaLinea* definito dal programmatore che disegna una linea i cui estremi hanno coordinate (x1, y1) e (x2, y2).

La linea è disegnata con una tonalità di grigio che risulterà più scura per valori crescenti della coordinata x1.

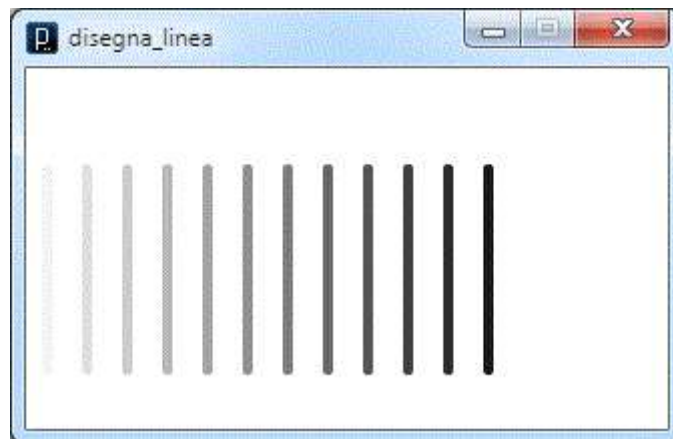
```
/* disegna_linea
   disegna linea con differenti tonalità di grigio */
/* istruzioni di inizializzazione del programma */
void setup() {
    size(320, 180);           // dimensione della finestra
    background(255);          // colore background bianco
    strokeWeight(5);           // spessore linea 5 pixel
    noLoop();
}
/* istruzioni di disegno nella finestra del programma; è
   eseguito solo una volta poiché il metodo noLoop() è
```



```

    invocato all'interno del metodo setup() */
void draw() {
    for(int x = 10; x<250; x=x+20){
        disegnaLinea(x, 50, x, 150);
    }
}
/* disegno linea con differenti tonalità di grigio */
void disegnaLinea(int x1, int y1, int x2, int y2){
    stroke(255-x1);
    line(x1, y1, x2, y2);
}

```



SINTASSI

Ogni riga che inizia con (//) è un commento.
 Ogni blocco compreso tra (/* */) è un commento.
 Ogni istruzione deve terminare con (;).
 Il linguaggio è case sensitive.

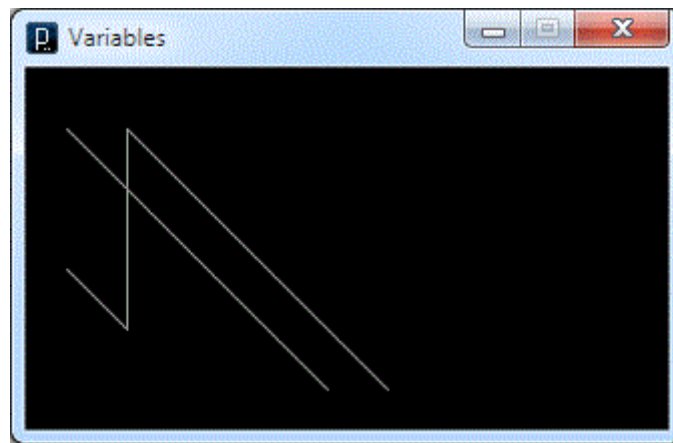
VARIABILI

Esempio.

```

size(320, 180);
background(0);
stroke(153);
int a = 20;
int b = 50;
int c = a*8;
int d = a*9;
int e = b-a;
int f = b*2;
int g = f+e;
line(a, f, b, g);
line(b, e, b, g);
line(b, e, d, c);
line(a, e, d-e, c);

```



Regole di visibilità delle variabili

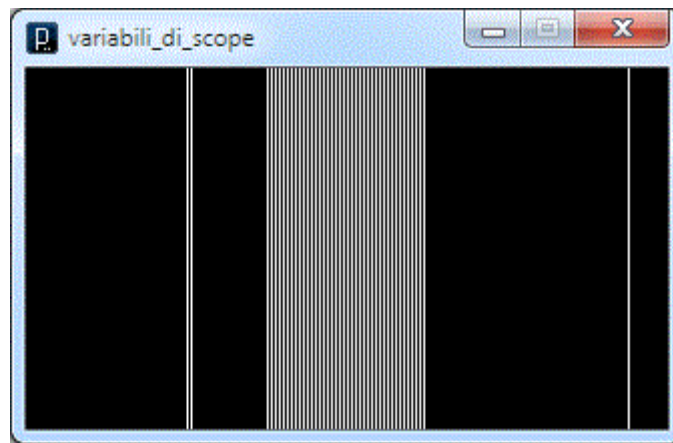
Una variabile locale è dichiarata all'interno di un metodo ed è visibile solo nel metodo in cui è dichiarata, la variabile è utilizzabile localmente al blocco in cui è utilizzata.

Una variabile globale è dichiarata al di fuori di ogni metodo ed è visibile in tutti i metodi.

Se due variabili hanno lo stesso nome, allora la variabile locale nasconde quella globale.

Esempio.

```
// a è una variabile globale
int a = 80;
void setup() {
  size(320, 180);
  background(0);
  stroke(255);
  noLoop();
}
void draw() {
  // disegna la linea usando la variabile globale a
  line(a, 0, a, height);
  // crea una nuova variabile a locale nel ciclo for
  for (int a = 120; a < 200; a += 2) {
    line(a, 0, a, height);
  }
  // disegna la linea usando la variabile locale a
  int a = 300;
  // disegna la linea usando una nuova variabile locale a
  line(a, 0, a, height);
  drawAnotherLine();
  drawYetAnotherLine();
}
void drawAnotherLine() {
  // crea una nuova variabile a local nel metodo
  int a = 320;
  // disegna la linea usando la variabile locale a
  line(a, 0, a, height);
}
void drawYetAnotherLine() {
  // usa la variabile globale a per disegnarne la linea
  line(a+2, 0, a+2, height);
}
```



TIPI DI DATI

int

Interi dell'insieme (-2.147.483.648..2.147.483.647).

long

Interi dell'insieme (-9.223.372.036.854.775.808..9.223.372.036.854.775.807).

byte

Interi dell'insieme (-128..127).

float

Numeri in virgola mobile a 32 bit dell'insieme (-3.40282347E+38..3.40282347E+38).

double

Numeri in virgola mobile a 64 bit.

boolean

Valori true e false.

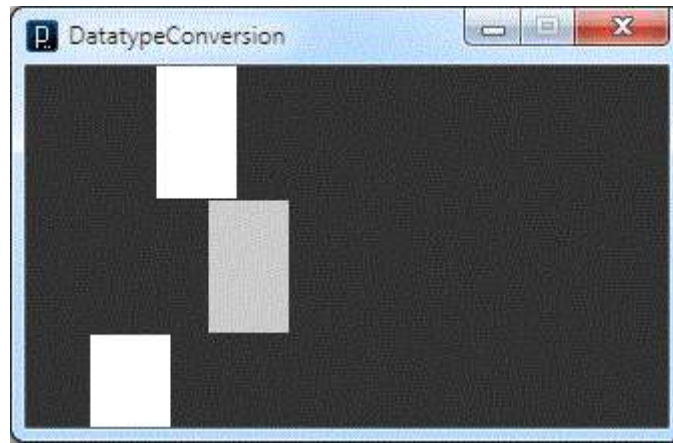
char

Caratteri delimitati da apici, ad esempio 'z'.

Esempio.

```
size(320, 180);
background(51);
noStroke();
char c;
float f;
int i;
byte b;
c = 'A';
f = float(c);           // f = 65.0
i = int(f * 1.4);
b = byte(c / 2);
rect(f, 0, 40, 66);
fill(204);
rect(i, 67, 40, 66);
fill(255);
```

```
rect(b, 134, 40, 66);
```

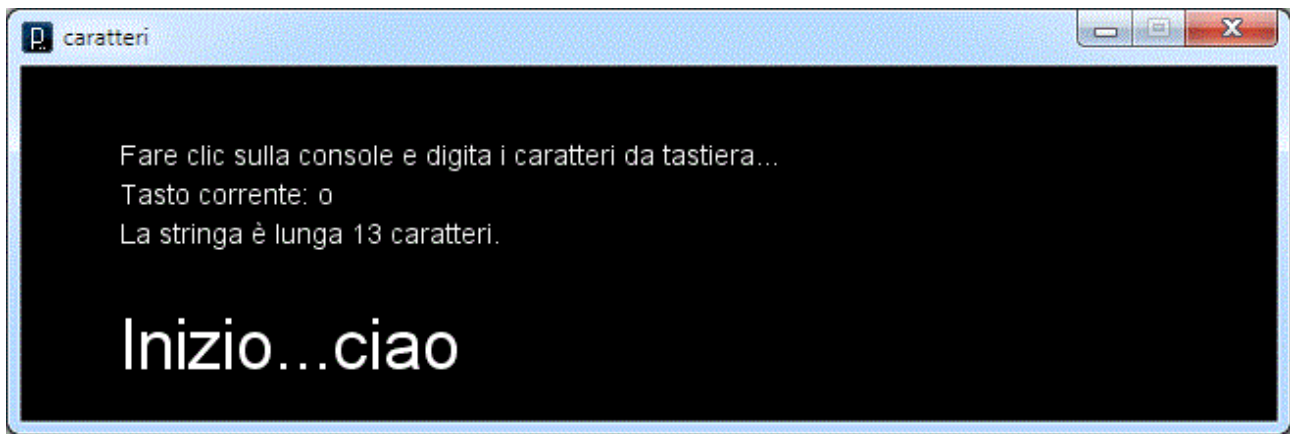


string

Sequenze di caratteri delimitati da doppi apici, ad esempio "Ciao mondo".

Esempio.

```
char letter;
String words = "Inizio...";
void setup() {
  size(640, 180);
  // crea il font
  textFont(createFont("Arial", 36));
}
void draw() {
  background(0);
  textSize(14);
  text("Fare clic sulla console e digita i caratteri da tastiera...", 50, 50);
  text("Tasto corrente: " + letter, 50, 70);
  text("La stringa è lunga " + words.length() + " caratteri.", 50, 90);
  textSize(36);
  text(words, 50, 120, 540, 300);
}
void keyPressed() {
  // la variabile "key" contiene il valore
  // del carattere digitato
  if ((key >= 'A' && key <= 'z') || key == ' ') {
    letter = key;
    words = words + key;
    // scrive il carattere sulla console
    println(key);
  }
}
```

Operatori aritmetici

Le operazioni aritmetiche di addizione, sottrazione, moltiplicazione, divisione e resto s'indicano rispettivamente con i simboli +, -, *, /, %.

L'operazione di resto (%) si applica solo a numeri interi, non a valori di tipo float e double.

Esempio.

```
int num = 20;
float c;
void setup()
{
  size(320,180);
  fill(255);
  frameRate(30);
}
void draw()
{
  background(0);
  c+=0.1;
  for(int i=1; i<height/num; i++) {
    float x = (c%i)*i*i;
    stroke(102);
    line(0, i*num, x, i*num);
    noStroke();
    rect(x, i*num-num/2, 8, num);
  }
}
```



Incremento e decremento

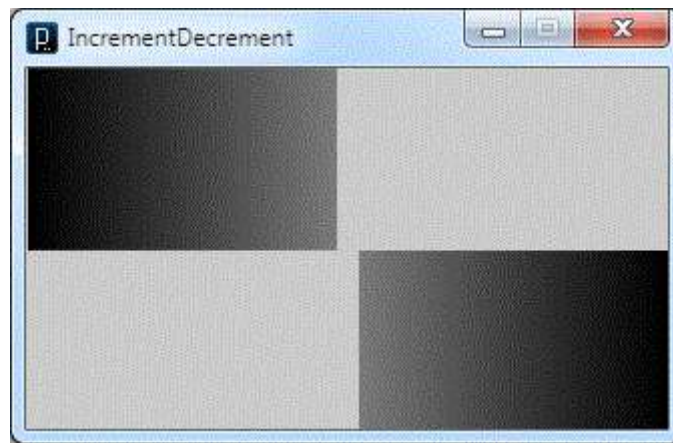
++ Aggiunge uno al valore della variabile.

-- Toglie uno dal valore della variabile.

Esempio.

```
int a;
int b;
boolean direction;
void setup()
{
  size(320, 180);
  colorMode(RGB, width);
  a = 0;
  b = width;
  direction = true;
  frameRate(30);
}
void draw()
{
  a++;
  if(a > width) {
    a = 0;
    direction = !direction;
  }
  if(direction == true){
    stroke(a);
  } else {
    stroke(width-a);
  }
  line(a, 0, a, height/2);

  b--;
  if(b < 0) {
    b = width;
  }
  if(direction == true) {
    stroke(width-b);
  } else {
    stroke(b);
  }
  line(b, height/2+1, b, height);
}
```



Operatori relazionali

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

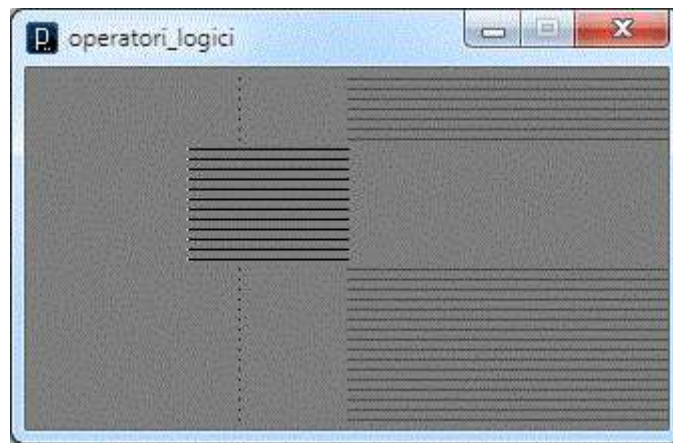
Operatori logici

&& AND.
 || OR.
 ! NOT.

Esempio.

```

size(320, 180);
background(126);
boolean test = false;
for (int i = 5; i <= height; i += 5) {
  // AND logico
  stroke(0);
  if((i > 35) && (i < 100)) {
    line(width/4, i, width/2, i);
    test = false;
  }
  // OR logico
  stroke(76);
  if ((i <= 35) || (i >= 100)) {
    line(width/2, i, width, i);
    test = true;
  }
  if (test) {
    stroke(0);
    point(width/3, i);
  }
  if (!test) {
    stroke(255);
    point(width/4, i);
  }
}
  
```



Precedenza degli operatori

Moltiplicativi: * / %

Additivi: + -

Relazionali: < > <= >=

Uguaglianza: == !=

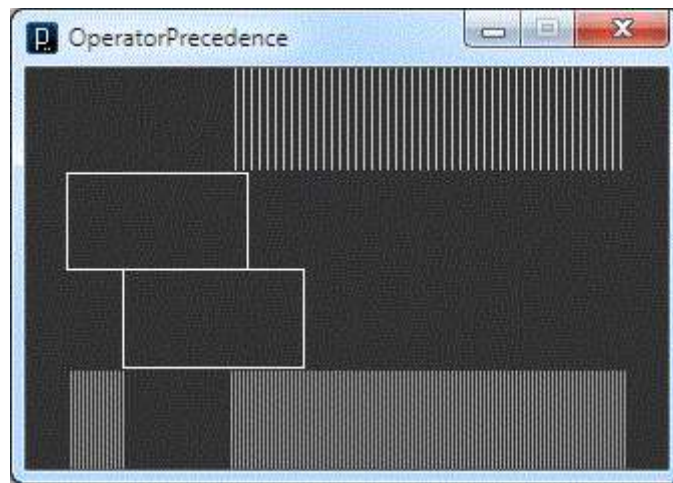
AND logico: &&

OR logico: ||

Assegnazione: = += -= *= /= %=

Esempio.

```
size(320, 200);
background(51);
noFill();
stroke(51);
stroke(204);
for(int i=0; i< width-20; i+= 4) {
  if(i > 30 + 70) {
    line(i, 0, i, 50);
  }
}
stroke(255);
rect(4 + 2 * 8, 52, 90, 48);
rect((4 + 2) * 8, 100, 90, 49);
stroke(153);
for(int i=0; i< width; i+= 2) {
  if(i > 20 && i < 50 || i > 100 && i < width-20) {
    line(i, 151, i, height-1);
  }
}
```

Concatenazione

Nel caso di valori di tipo String, il (+) concatena due stringhe; consente anche di concatenare una stringa con un numero.

Esempio.

```
int anno=1492, mese=10, giorno=12;
print("Data scoperta dell'America: ");
println(giorno + "/" + mese + "/" + anno);
```

Variabili di sistema

frameCount

È automaticamente incrementato ogni volta che il metodo *draw* è eseguito e vale zero quando si esegue il metodo *setup*.

Esempio.

```
void draw() {
  println(frameCount);
}
```

Stampa sulla console il numero di frame, partendo da uno, infinitamente fino a quando s'interrompe l'esecuzione del programma.

screen

Memorizza le dimensioni dello schermo del PC alla risoluzione correntemente impostata. Ad esempio, se la risoluzione dello schermo è 1024X768, *screen.width* vale 1024 e *screen.height* vale 768.

Esempio.

```
void setup() {
  size(screen.width, screen.height);
}
void draw() {
  line(0, 0, screen.width, screen.height);
}
```

Stampa una linea dalla coordinata in alto a sinistra (0;0) fino alla coordinata in basso a destra (1024;768).

width

Memorizza la larghezza della finestra corrente, prendendo tale valore dal primo parametro della funzione *size*.

Ad esempio, se è richiamato il metodo *size(320, 240)*, *width* vale 320.

height

Memorizza l'altezza della finestra corrente, prendendo tale valore dal secondo parametro della funzione *size*.

Ad esempio, se è richiamato il metodo *size(320, 240)*, *height* vale 240

STRUTTURE DI CONTROLLO

SEQUENZA

// istruzione vuota

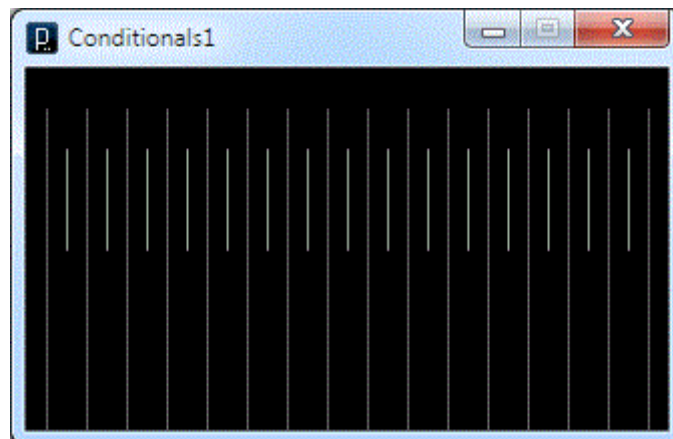
SELEZIONE

Selezione unaria - Selezione binaria

Esempio.

Esempio.

```
size(320, 180);  
background(0);  
for(int i=10; i<width; i+=10) {  
  if(i%20 == 0) {  
    stroke(153);  
    line(i, 40, i, height/2);  
  } else {  
    stroke(102);  
    line(i, 20, i, 180);  
  }  
}
```



Selezione nidificata

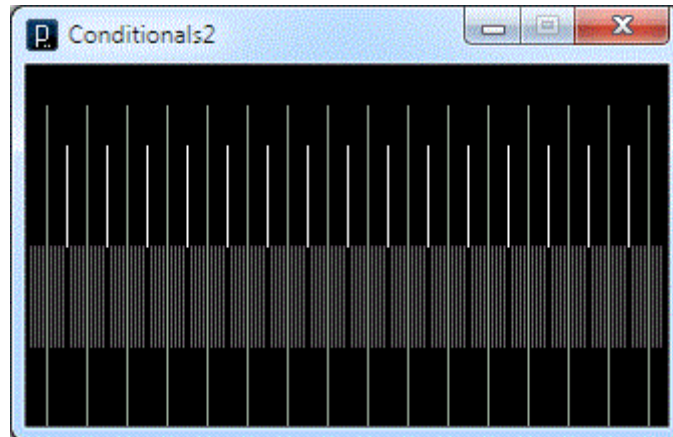
Esempio.

```
size(320, 180);  
background(0);  
for(int i=2; i<width-2; i+=2) {  
  if(i%20 == 0) {  
    stroke(255);  
    line(i, 40, i, height/2);  
  } else if (i%10 == 0) {
```

```

    stroke(153);
    line(i, 20, i, 180);
  } else {
    stroke(102);
    line(i, height/2, i, height-40);
  }
}

```



Selezione multipla

Esempio.

```

char letter = 'B';
switch(letter) {
  case 'A':
    println("Lettera A");           // non è eseguita
    break;
  case 'B':
    println("Lettera B");           // stampa "Lettera B"
    break;
  default:
    println("Altre lettere");       // non è eseguita
    break;
}

```

ITERAZIONE

Il numero d'iterazioni non è noto a priori

Ciclo a condizione iniziale

Esempio.

```

int i = 0;
while (i < 10)
{
  println("i = " + i);
  i++;
}

```


Ciclo a condizione finale

Il numero d'iterazioni è noto a priori

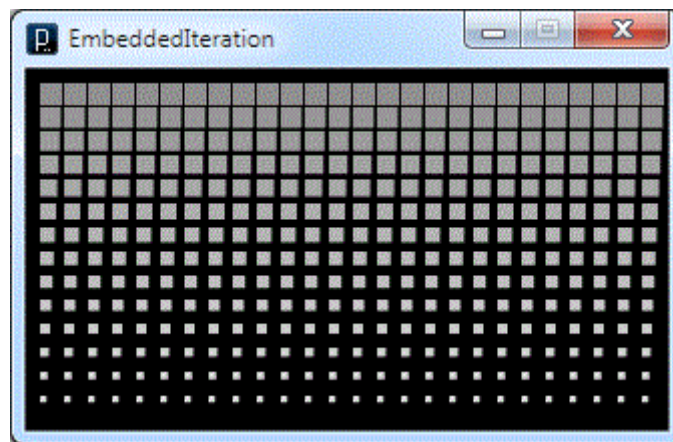
Ciclo enumerativo

Esempio.

```
for (int i = 0; i < 10; i++)  
    println("i = "+ i);
```

Esempio.

```
float box_size = 11;  
float box_space = 12;  
int margin = 7;  
size(200, 200);  
background(0);  
noStroke();  
for (int i = margin; i < height-margin; i += box_space){  
    if(box_size > 0){  
        for(int j = margin; j < width-margin; j+= box_space){  
            fill(255-box_size*10);  
            rect(j, i, box_size, box_size);  
        }  
        box_size = box_size - 0.6;  
    }  
}
```



exit

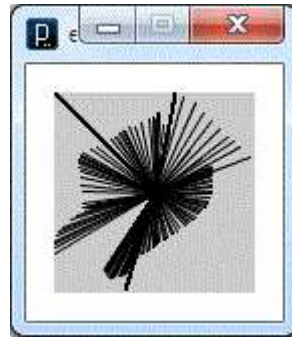
Termina l'esecuzione del programma.

I programmi senza la funzione *draw* si fermano automaticamente dopo l'esecuzione dell'ultima linea, mentre i programmi che usano la funzione *draw* continuano l'esecuzione fino a quando il programma è fermato manualmente tramite il pulsante **Stop** della toolbar oppure è eseguito il metodo *exit* durante il run-time.

Esempio.

```
/* exit_01  
   disegna delle linee concentriche seguendo il movimento del mouse  
   fino alla pressione del tasto ESC */  
void draw() {
```

```
line(mouseX, mouseY, 50, 50);  
if ( keyPressed && key == 27 )  
    exit();  
}
```



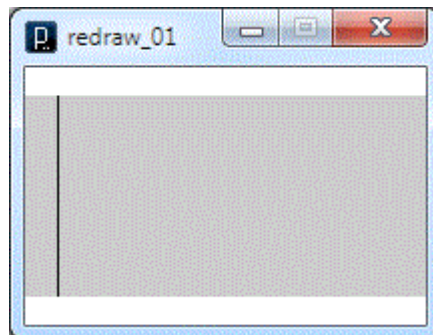
FUNZIONI

redraw

Permette al programma di aggiornare la finestra di display eseguendo il metodo *draw* solo se necessario, ad esempio quando si verifica un evento gestito dai metodi *mousePressed* o *keyPressed*.

Esempio.

```
/* redraw_01
   disegna una linea verticale che avanza verso destra
   ad ogni pressione di un qualsiasi pulsante del mouse */
float x = 0;
void setup() {
  size(200, 100);
  noLoop();
}
void draw() {
  background(204);
  line(x, 0, x, height);
}
void mousePressed() {
  x++;
  redraw();
}
```



Esempio.

```
/* classe_04 programma principale
   I cerchi vengono memorizzati in un array di oggetti;
   ad ogni click del mouse si crea un nuovo cerchio che
   viene aggiunto all'array e disegnato sulla finestra
   di disegno, fino ad un massimo di 10 cerchi */
Circle [] c ;
int i;
void setup(){
  size(320,180);
  background(255,255,255);
  smooth();
  c= new Circle[10];
  i=-1;
```

```

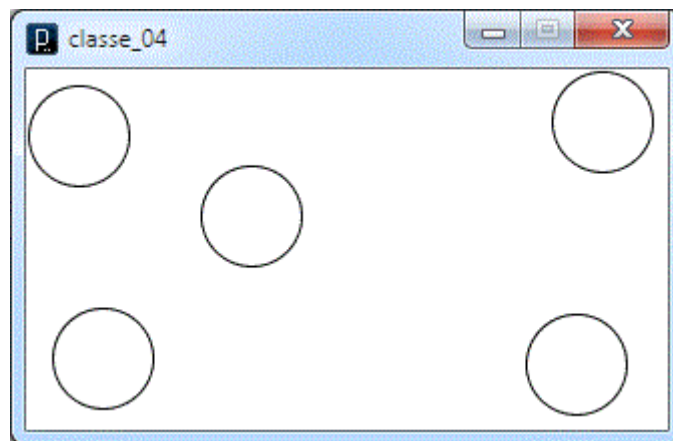
    noLoop();
}
void draw(){
    if ((0<=i) && (i<10))
        c[i].display();
    }
void mouseClicked(){
    i=i+1;
    if (i<10){
        c[i]=new Circle(mouseX, mouseY, 50);
        redraw();
    };
}

/* Circle classe Circle */
class Circle {
    int x, y, d;

    Circle(int cx, int cy, int diam){
        x=cx;
        y=cy;
        d=diam;
    }

    void display (){
        ellipse(x, y, d,d);
    }
}

```



abs

Restituisce il valore assoluto di un numero.

Il valore assoluto di un numero è il numero privo del segno.

Esempio.

```

int a = abs(153);           // pone a pari a 153
int b = abs(-15);          // pone b pari a 15
float c = abs(12.234);      // pone c pari a 12.234
float d = abs(-9.23);       // pone d pari a 9.23

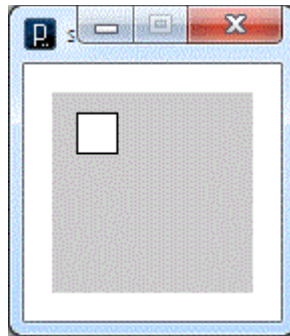
```


constrain(valore, minimo, massimo)

Costringe valore a non eccedere il valore minimo e massimo impostato.

Esempio.

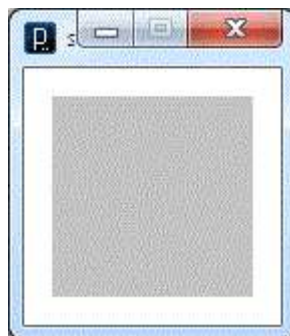
```
/* constrain_01  
   disegna un rettangolo che si sposta con il movimento del  
   mouse nell'ambito delle coordinate fissate da constrain */  
void draw() {  
  background(204);  
  float mx = constrain(mouseX, 10, 70);  
  float my = constrain(mouseY, 10, 70);  
  rect(mx, my, 20, 20);  
}
```

**dist(x1, y1, x2, y2)**

Calcola la distanza tra due punti.

Esempio.

```
/* dist_01  
   cambia il colore di background in base alla distanza  
   del mouse dal centro */  
void draw() {  
  float d = dist(50, 50, mouseX, mouseY);  
  background(d*4);  
}
```

**max(valore1, valore2)****max(valore1, valore2, valore3)****max(array)**

Calcola il massimo valore di una sequenza di numeri.

Esempio.

```
int a = max(5, 9);           // pone a pari a 9
int b = max(-4, -12);        // pone b pari a -4
float c = max(12.3, 230.24); // pone c pari a 230.24
int[] list = { 9, -4, 230, -20, 40 };
int h = max(list);           // pone h pari a 230
```

min(valore1, valore2)
min(valore1, valore2, valore3)
min(array)

Calcola il minimo valore di una sequenza di numeri.

Esempio.

```
int a = min(5, 9);           // pone a pari a 5
int b = min(-4, -12);        // pone b pari a -12
float c = min(12.3, 230.24); // pone c pari a 12.3
int[] list = { 9, -4, 230, -20, 40 };
int h = min(list);           // pone h pari a -20
```

pow(base, potenza)

Restituisce una base elevata a potenza.

Esempio.

```
float a = pow( 1, 3);         // pone a pari a 1*1*1 = 1
float b = pow( 3, 5);         // pone b pari a 3*3*3*3*3 = 243
float c = pow( 3,-5);         // pone c pari a 1 / 3*3*3*3*3 = 1 / 243 = .0041
float d = pow(9, 0.5);        // calcola la radice quadrata di 9; pone d = 3
```

sqrt(numero)

Calcola la radice quadrata di un numero.

Esempio.

```
float b = sqrt(25);           // pone a pari a 5
```

sin(angle) - cos(ang1)

Seno e coseno.

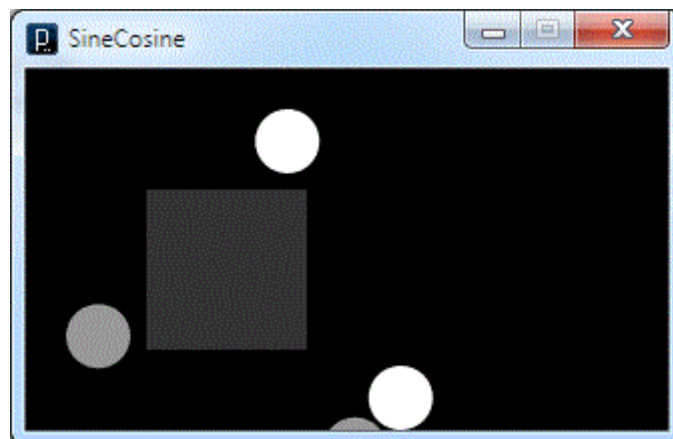
Esempio.

```
int i = 45;
int j = 225;
float pos1 = 0;
float pos2 = 0;
float pos3 = 0;
float pos4 = 0;
int sc = 40;
void setup()
{
  size(320, 180);
  noStroke();
```

```

    smooth();
}
void draw()
{
    background(0);
    fill(51);
    rect(60, 60, 80, 80);
    fill(255);
    ellipse(pos1, 36, 32, 32);
    fill(153);
    ellipse(36, pos2, 32, 32);
    fill(255);
    ellipse(pos3, 164, 32, 32);
    fill(153);
    ellipse(164, pos4, 32, 32);
    i += 3;
    j -= 3;
    if(i > 405) {
        i = 45;
        j = 225;
    }
    float ang1 = radians(i); // convert degrees to radians
    float ang2 = radians(j); // convert degrees to radians
    pos1 = width/2 + (sc * cos(ang1));
    pos2 = width/2 + (sc * sin(ang1));
    pos3 = width/2 + (sc * cos(ang2));
    pos4 = width/2 + (sc * sin(ang2));
}

```



atan2(my-ey, mx-ex);
Arcotangente.

Esempio.

```

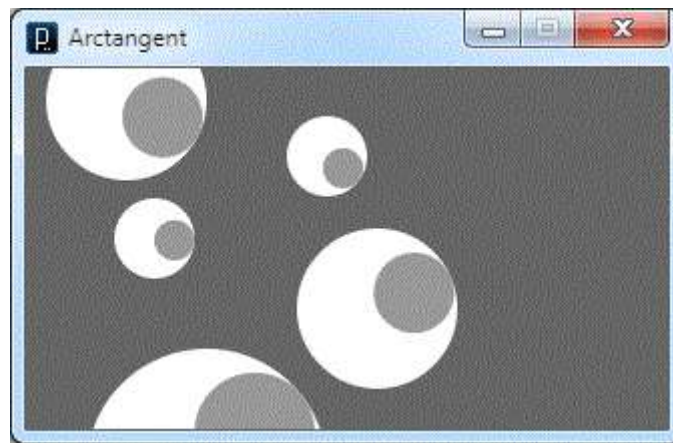
Eye e1, e2, e3, e4, e5;
void setup()
{
    size(320, 180);
    smooth();
    noStroke();
    e1 = new Eye( 50, 16, 80);

```

```

e2 = new Eye( 64, 85, 40);
e3 = new Eye( 90, 200, 120);
e4 = new Eye(150, 44, 40);
e5 = new Eye(175, 120, 80);
}
void draw()
{
  background(102);
  e1.update(mouseX, mouseY);
  e2.update(mouseX, mouseY);
  e3.update(mouseX, mouseY);
  e4.update(mouseX, mouseY);
  e5.update(mouseX, mouseY);
  e1.display();
  e2.display();
  e3.display();
  e4.display();
  e5.display();
}
class Eye
{
  int ex, ey;
  int size;
  float angle = 0.0;
  Eye(int x, int y, int s) {
    ex = x;
    ey = y;
    size = s;
  }
  void update(int mx, int my) {
    angle = atan2(my-ey, mx-ex);
  }
  void display() {
    pushMatrix();
    translate(ex, ey);
    fill(255);
    ellipse(0, 0, size, size);
    rotate(angle);
    fill(153);
    ellipse(size/4, 0, size/2, size/2);
    popMatrix();
  }
}

```



Metodi

Distinguiamo due tipi di metodi.

1. Procedure: eseguono delle istruzioni producendo come effetto la stampa e/o il disegno di dati, la modifica di variabili; si riconoscono perché nell'intestazione compare la parola *void* prima del nome del metodo.
2. Funzioni: oltre a eseguire istruzioni come le procedure, restituiscono sempre un valore del tipo specificato nell'intestazione del metodo, ad esempio *int* se il metodo restituisce un valore intero; il valore restituito da una funzione può essere stampato o assegnato ad una variabile per un uso successivo.

Esempio, procedura che calcola la somma dei primi 10 numeri.

```
void setup()
{
  int n = 10;
  T(n);
}
void T(int x)
{
  int s=0, i;
  for ( i=1; i<=x; i++)
    s+=i;
  print("La somma dei primi " + x + " numeri e' "+s);
}
```

Esempio, funzione che calcola la somma dei primi 10 numeri.

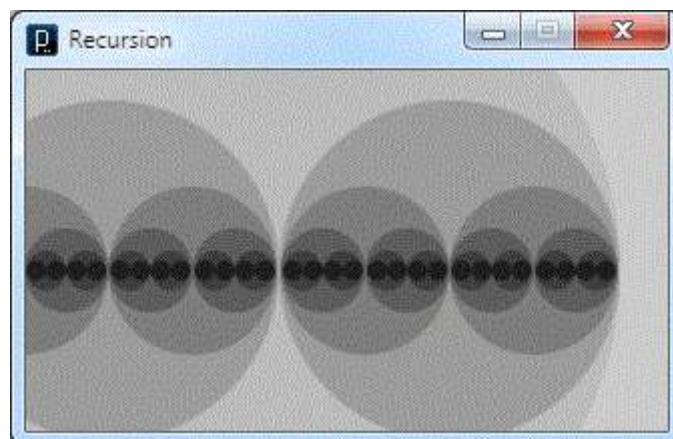
```
void setup()
{
  int n=10;
  print("La somma dei primi " + n + " numeri e' "+ T(n));
}
int T(int x)
{
  int s=0, i;
  for ( i=1; i<=x; i++)
    s+=i;
  return s;
}
```

La somma dei primi 10 numeri e' 55

RICORSIONE

Esempio.

```
void setup()
{
  size(320, 180);
  noStroke();
  smooth();
  noLoop();
}
void draw()
{
  drawCircle(126, 170, 6);
}
void drawCircle(int x, int radius, int level)
{
  float tt = 126 * level/4.0;
  fill(tt);
  ellipse(x, 100, radius*2, radius*2);
  if(level > 1) {
    level = level - 1;
    drawCircle(x - radius/2, radius/2, level);
    drawCircle(x + radius/2, radius/2, level);
  }
}
```



Esempio, disegnare un albero.

```
float theta;
void setup() {
  size(640, 360);
  smooth();
}
```

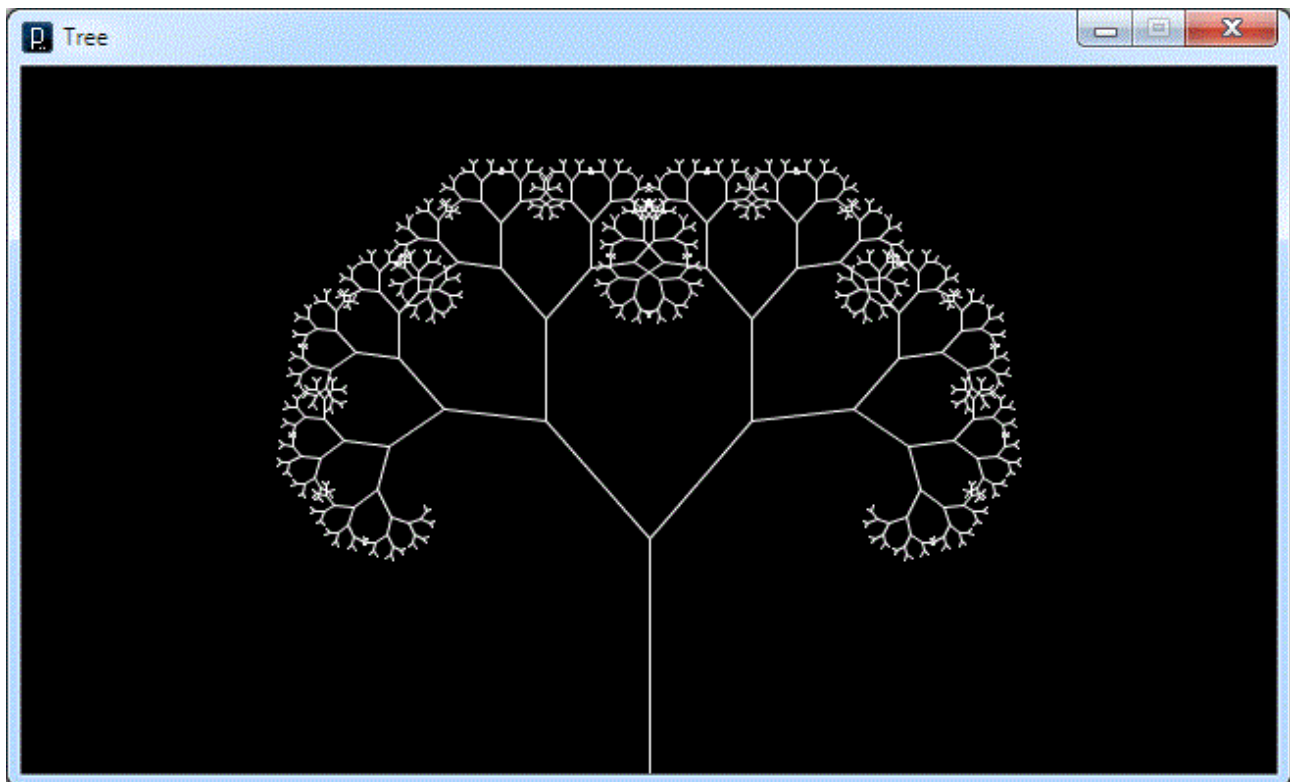


```

void draw() {
  background(0);
  frameRate(30);
  stroke(255);
  float a = (mouseX / (float) width) * 90f;
  // conversione in radianti
  theta = radians(a);
  translate(width/2,height);
  // disegna linee di 120 pixels
  line(0,0,0,-120);
  translate(0,-120);
  branch(120);
}

void branch(float h) {
  h *= 0.66;
  if (h > 2) {
    pushMatrix();
    rotate(theta);
    line(0, 0, 0, -h);
    translate(0, -h);
    branch(h);
    popMatrix();
    pushMatrix();
    rotate(-theta);
    line(0, 0, 0, -h);
    translate(0, -h);
    branch(h);
    popMatrix();
  }
}

```



ARRAY

INTRODUZIONE

Un array è una sequenza di oggetti di tipo omogeneo, ognuno dei quali può essere identificato da uno o più indici numerici.

Il primo elemento di un array ha indice zero, l'ultimo ($n-1$), dove n è il numero di elementi dell'array.

Gli array prima di essere usati vanno dichiarati e poi allocati.

Esempio.

```
int [ ] a ;           // dichiarazione di un array di int
a = new int [10] ;    // allocazione di 10 elementi contenenti int
```

Gli array possono essere dichiarati e allocati contemporaneamente.

```
int a [ ] = new int [10]
```

Oppure possono essere dichiarati e inizializzati.

```
int b [ ] = {3, -5, -2}
```

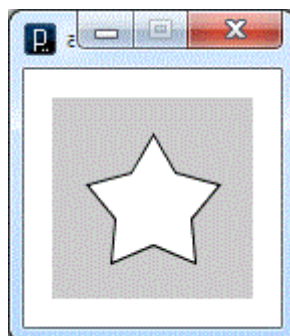
length

Ritorna la lunghezza dell'array.

```
print(a.length )
// stampa la lunghezza, ovvero il numero di componenti del vettore a: 10
```

Esempio, disegnare una stella.

```
/* array_01
   disegna una stella */
int x[] = { 50,61,83,69,71,50,29,31,17,39 };
int y[] = { 18,37,43,60,82,73,82,60,43,37 };
beginShape();
for (int i=0;i<10;i++)
    vertex(x[i],y[i]);
endShape(CLOSE);
```



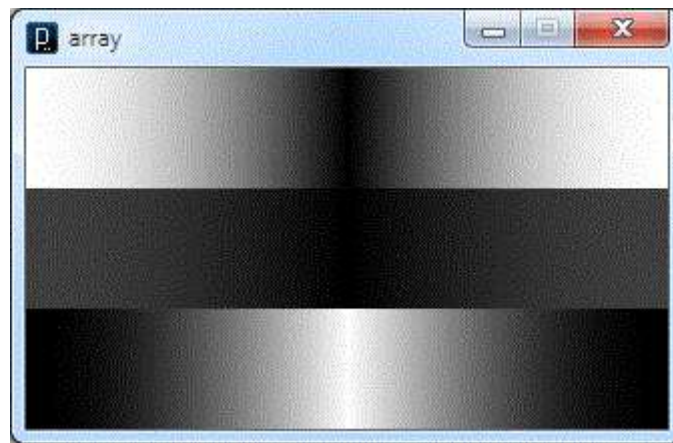
Esempio, stampare sulla console dieci numeri estratti casualmente.

```
/* array_02
   estrazione numeri random */
int MAX = 10;
int[] tabella = new int[MAX];
for (int i = 0; i < MAX; i++)
    tabella[i] = int(random(1,7)); // estrae numero random tra 1 e 6
println(tabella.length + " elementi:");
println(tabella);
```

```
10 elementi:
[0] 6
[1] 3
[2] 6
[3] 5
[4] 4
[5] 2
[6] 2
[7] 3
[8] 1
[9] 2
```

Esempio.

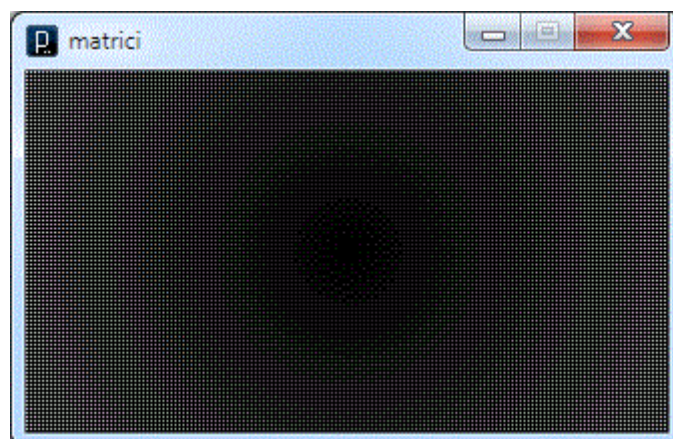
```
size(200, 200);
float[] coswave = new float[width];
for (int i = 0; i < width; i++) {
    float amount = map(i, 0, width, 0, PI);
    coswave[i] = abs(cos(amount));
}
for (int i = 0; i < width; i++) {
    stroke(coswave[i]*255);
    line(i, 0, i, height/3);
}
for (int i = 0; i < width; i++) {
    stroke(coswave[i]*255 / 4);
    line(i, height/3, i, height/3*2);
}
for (int i = 0; i < width; i++) {
    stroke(255 - coswave[i]*255);
    line(i, height/3*2, i, height);
}
```



MATRICI

Esempio.

```
float[][] distances;
float maxDistance;
size(320, 180);
background(0);
maxDistance = dist(width/2, height/2, width, height);
distances = new float[width][height];
for(int i=0; i<height; i++) {
  for(int j=0; j<width; j++) {
    float dist = dist(width/2, height/2, j, i);
    distances[j][i] = dist/maxDistance * 255;
  }
}
for(int i=0; i<height; i+=2) {
  for(int j=0; j<width; j+=2) {
    stroke(distances[j][i]);
    point(j, i);
  }
}
```



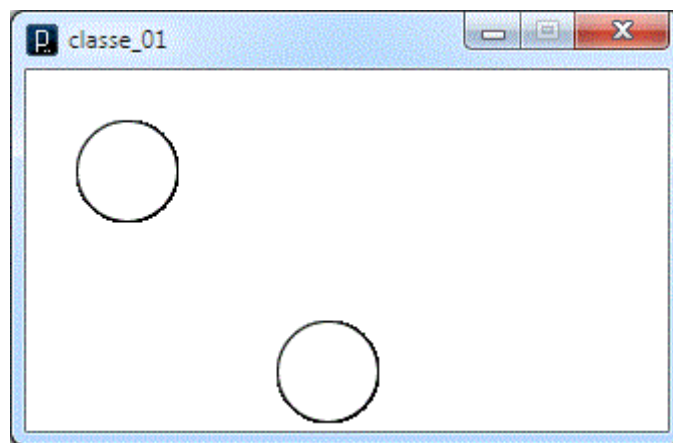
OOP (*OBJECT ORIENTED PROGRAMMING*)

INTRODUZIONE

Processing è un linguaggio orientato agli oggetti.

Esempio, programmazione imperativa.

```
/* classe_01
   Disegno cerchi senza utilizzare la programmazione OO */
int xa, ya, da;
int xb, yb, db;
void setup(){
  size(320,180);
  background(255,255,255);
  smooth();
  xa=50;
  ya=50;
  da=50;
  xb=150;
  yb=150;
  db=50;
}
void draw(){
  ellipse(xa,ya,da,da);
  ellipse(xb,yb,db,db);
}
```



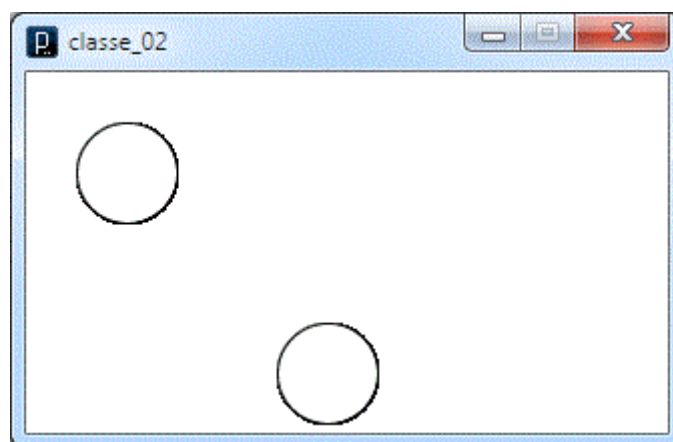
Esempio, programmazione OO.

```
/* classe_02 programma principale
   Disegno cerchi utilizzando la programmazione OO */
Circle ca, cb;
void setup(){
  size(320,180);
  background(255,255,255);
  smooth();
  ca= new Circle(50, 50, 50);
  cb= new Circle(150, 150, 50);
}
```

```

}
void draw(){
  ca.display();
  cb.display();
}
/* Circle classe Circle */
class Circle {
  int x, y, d;
  Circle(int cx, int cy, int diam){
    x=cx;
    y=cy;
    d=diam;
  }
  void display (){
    ellipse(x, y, d,d);
  }
}

```



Esempio, agli attributi di un oggetto si accede singolarmente come mostra questo programma che disegna due file di cerchi.
Nelle due righe di codice.

```

ca.x = ca.x +50;
cb.x = cb.x +50;

```

L'espressione `ca.x` indica l'attributo `x` dell'oggetto `ca` mentre `cb.x` indica l'attributo `x` dell'oggetto `cb`.

Esempio.

```

/* classe_03 programma principale
   Disegno cerchi utilizzando la programmazione OO */
Circle ca, cb;
void setup(){
  size(400,300);
  background(255,255,255);
  smooth();
  ca= new Circle(50, 150, 50);
  cb= new Circle(150, 250, 50);
  frameRate(3);
}

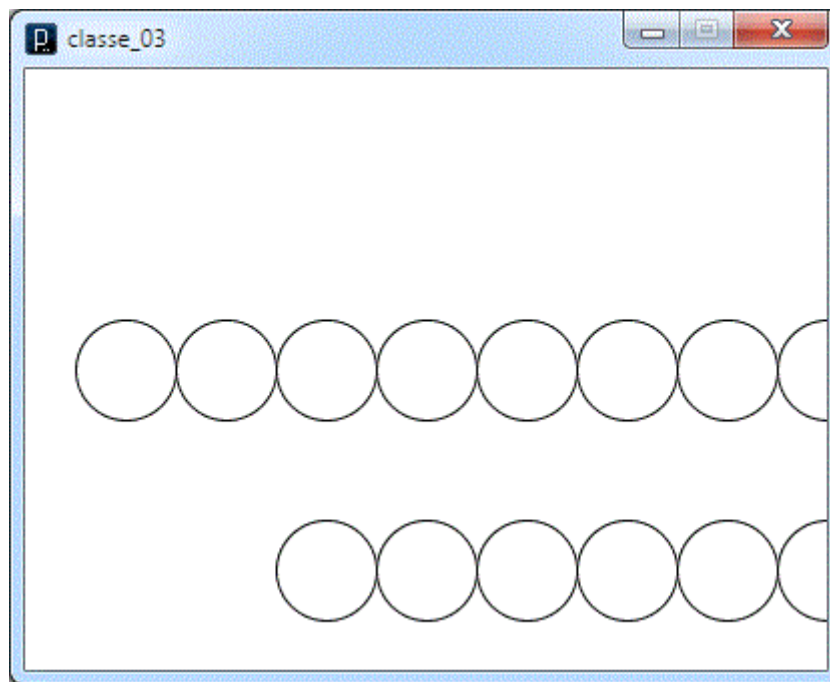
```



```

void draw(){
  ca.display();
  cb.display();
  if (ca.x < 425) {
    ca.x = ca.x +50;
    cb.x = cb.x +50;
  }
}
/* Circle classe Circle */
class Circle {
  int x, y, d;
  Circle(int cx, int cy, int diam){
    x=cx;
    y=cy;
    d=diam;
  }
  void display (){
    ellipse(x, y, d,d);
  }
}

```



Esempio.

```

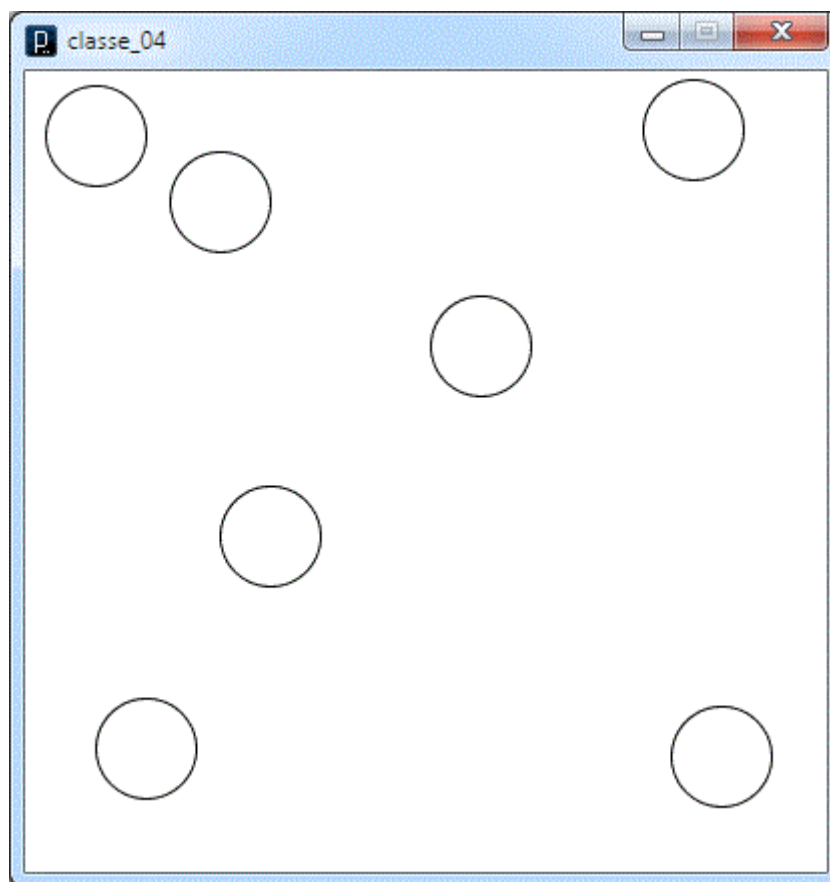
/* classe_04 programma principale
  I cerchi sono memorizzati in un array di oggetti;
  ad ogni click del mouse si crea un nuovo cerchio che
  viene aggiunto all'array e disegnato sulla finestra
  di disegno, fino ad un massimo di 10 cerchi */
Circle [] c ;
int i;
void setup(){
  size(400,400);
  background(255,255,255);
  smooth();

```

```

c= new Circle[10];
i=-1;
noLoop();
}
void draw(){
  if ((0<=i) && (i<10))
    c[i].display();
  }
void mouseClicked(){
  i=i+1;
  if (i<10){
    c[i]=new Circle(mouseX, mouseY, 50);
    redraw();
  };
}
/* Circle classe Circle */
class Circle {
  int x, y, d;
  Circle(int cx, int cy, int diam){
    x=cx;
    y=cy;
    d=diam;
  }
  void display (){
    ellipse(x, y, d,d);
  }
}

```



EREDITARIETÀ

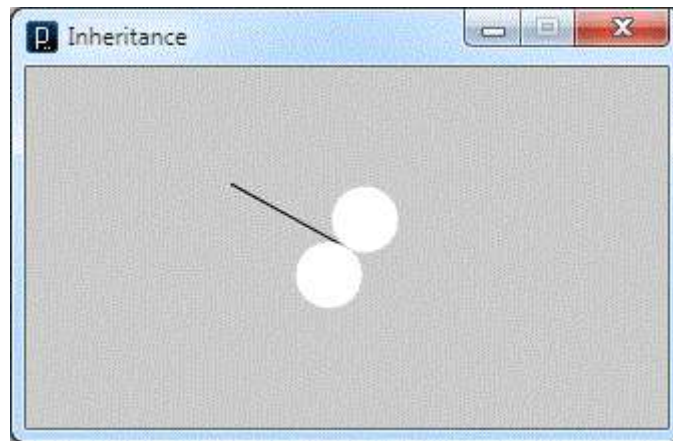
Esempio.

```
SpinSpots spots;
SpinArm arm;
void setup()
{
  size(320, 180);
  smooth();
  arm = new SpinArm(width/2, height/2, 0.01);
  spots = new SpinSpots(width/2, height/2, -0.02, 33.0);
}
void draw()
{
  background(204);
  arm.update();
  arm.display();
  spots.update();
  spots.display();
}
class Spin
{
  float x, y, speed;
  float angle = 0.0;
  Spin(float xpos, float ypos, float s) {
    x = xpos;
    y = ypos;
    speed = s;
  }
  void update() {
    angle += speed;
  }
}
class SpinArm extends Spin
{
  SpinArm(float x, float y, float s) {
    super(x, y, s);
  }
  void display() {
    strokeWeight(1);
    stroke(0);
    pushMatrix();
    translate(x, y);
    angle += speed;
    rotate(angle);
    line(0, 0, 66, 0);
    popMatrix();
  }
}
class SpinSpots extends Spin
{
  float dim;
```

```

SpinSpots(float x, float y, float s, float d) {
  super(x, y, s);
  dim = d;
}
void display() {
  noStroke();
  pushMatrix();
  translate(x, y);
  angle += speed;
  rotate(angle);
  ellipse(-dim/2, 0, dim, dim);
  ellipse(dim/2, 0, dim, dim);
  popMatrix();
}
}

```



OVERRIDING

Esempio.

```

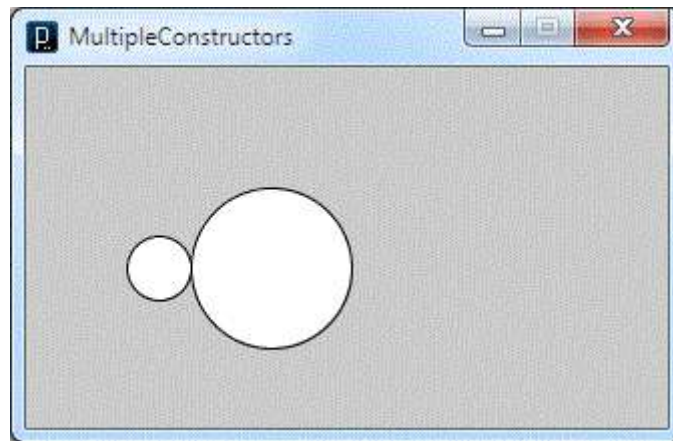
Spot sp1, sp2;
void setup()
{
  size(320, 180);
  background(204);
  smooth();
  noLoop();
  // run del costruttore senza parametri
  sp1 = new Spot();
  // run del costruttore con tre parametri
  sp2 = new Spot(122, 100, 40);
}
void draw() {
  sp1.display();
  sp2.display();
}
class Spot {
  float x, y, radius;
  // prima versione del costruttore di Spot
  // ai campi sono assegnati i valori di default

```

```

Spot() {
  x = 66;
  y = 100;
  radius = 16;
}
// seconda versione del costruttore di Spot
Spot(float xpos, float ypos, float r) {
  x = xpos;
  y = ypos;
  radius = r;
}
void display() {
  ellipse(x, y, radius*2, radius*2);
}
}

```



ARRAY DI OGGETTI

Esempio.

```

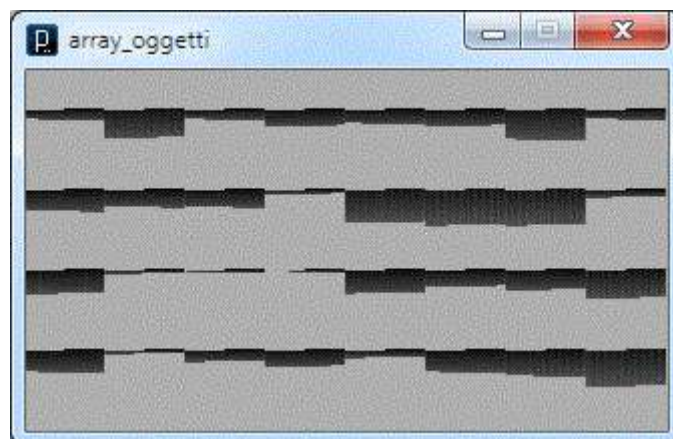
int unit = 40;
int count;
Module[] mods;
void setup() {
  size(320, 180);
  background(176);
  noStroke();
  int wideCount = width / unit;
  int highCount = height / unit;
  count = wideCount * highCount;
  mods = new Module[count];
  int index = 0;
  for (int y = 0; y < highCount; y++) {
    for (int x = 0; x < wideCount; x++) {
      mods[index++] = new Module(x*unit, y*unit, unit/2, unit/2, random(0.05, 0.8));
    }
  }
}
void draw() {
  for (int i = 0; i < count; i++) {

```

```

    mods[i].update();
    mods[i].draw();
  }
}
class Module {
  int mx, my;
  int big;
  float x, y;
  int xdir = 1;
  int ydir = 1;
  float speed;
  // costruttore
  Module(int imx, int imy, int ix, int iy, float ispeed) {
    mx = imx;
    my = imy;
    x = ix;
    y = iy;
    speed = ispeed;
    big = unit;
  }
  void update() {
    x = x + (speed * xdir);
    if (x >= big || x <= 0) {
      xdir *= -1;
      x = x + (1 * xdir);
      y = y + (1 * ydir);
    }
    if (y >= big || y <= 0) {
      ydir *= -1;
      y = y + (1 * ydir);
    }
  }
  void draw() {
    stroke(second() * 4);
    point(mx+x-1, my+y-1);
  }
}
}

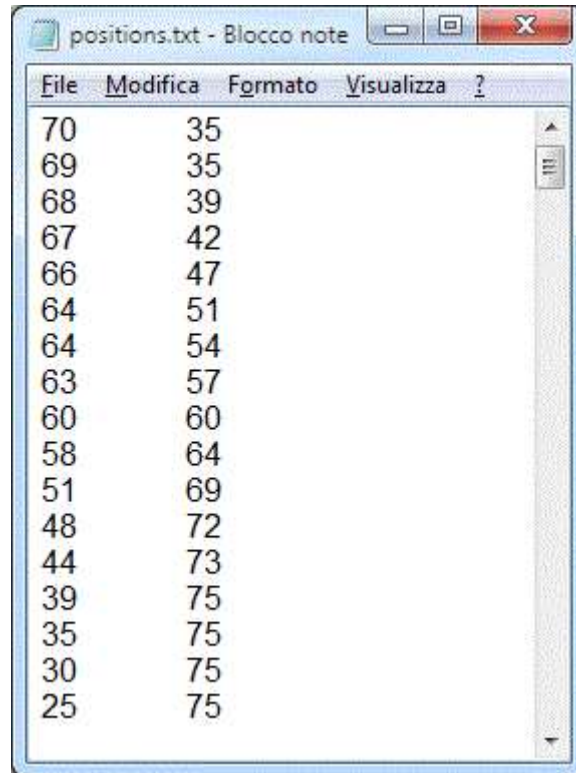
```



FILE

APRI FILE

Esempio, aprire un file di testo che contiene due colonne di numeri separati da TAB (t).



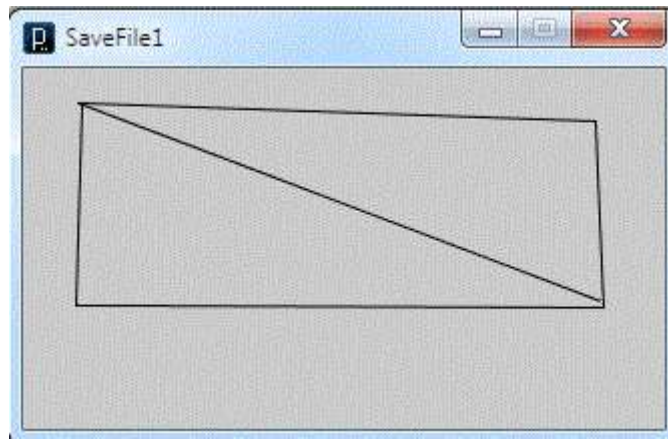
```
String[] lines;
int index = 0;
void setup() {
  size(320, 180);
  background(0);
  stroke(255);
  frameRate(12);
  lines = loadStrings("positions.txt");
}
void draw() {
  if (index < lines.length) {
    String[] pieces = split(lines[index], '\t');
    if (pieces.length == 2) {
      int x = int(pieces[0]) * 2;
      int y = int(pieces[1]) * 2;
      point(x, y);
    }
    index = index + 1;
  }
}
```



SALVA FILE

Esempio, salvare un file di testo che contiene due colonne di numeri separati da TAB (`\t`). Il metodo `saveStrings` scrive un array di stringhe in un file di testo; ogni stringa è scritta in una nuova linea.

```
int[] x = new int[0];
int[] y = new int[0];
void setup()
{
  size(200, 200);
}
void draw()
{
  background(204);
  stroke(0);
  noFill();
  beginShape();
  for (int i = 0; i < x.length; i++) {
    vertex(x[i], y[i]);
  }
  endShape();
  if (x.length >= 1) {
    stroke(255);
    line(mouseX, mouseY, x[x.length-1], y[x.length-1]);
  }
}
void mousePressed() {
  x = append(x, mouseX);
  y = append(y, mouseY);
}
void keyPressed() {
  String[] lines = new String[x.length];
  for (int i = 0; i < x.length; i++) {
    lines[i] = x[i] + "\t" + y[i];
  }
  saveStrings("lines.txt", lines);
  exit();
}
```

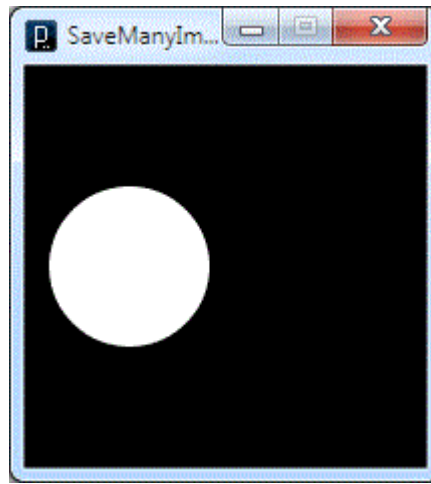


Quando si preme un tasto qualsiasi i dati sono salvati nel file DATI.TXT.

File	Modifica	Formato	Visualizza	?
36	28			
36	28			
36	28			
34	148			
34	148			
175	148			
175	148			
171	31			
171	31			
37	27			
37	27			
171	144			

Esempio, salvare i primi 50 frame del programma.
Il metodo `saveFrame` salva le immagini quando il programma è in run.

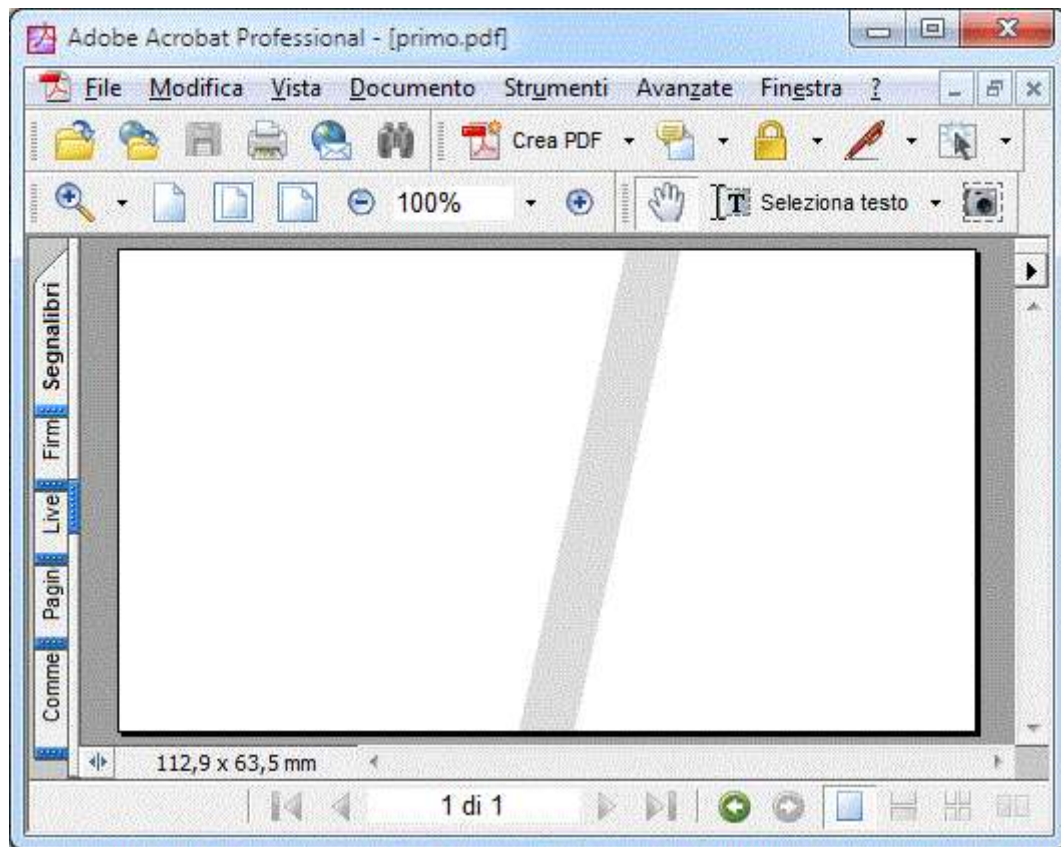
```
float x = 33;
float numFrames = 50;
void setup()
{
  size(200, 200);
  smooth();
  noStroke();
}
void draw()
{
  background(0);
  x += random(-2, 2);
  ellipse(x, 100, 80, 80);
  if (frameCount <= numFrames) {
    saveFrame("cerchi-####.tif");
  }
}
```



PDF (*PORTABLE DOCUMENT FORMAT*)

Esempio, salvare un frame come file **PDF**.

```
import processing.pdf.*;
void setup()
{
  size(320, 180, PDF, "primo.pdf");
}
void draw()
{
  background(255);
  stroke(0, 20);
  strokeWeight(20.0);
  line(200, 0, width/2, height);
  exit();
}
```



Esempio, premere il carattere (s) per salvare la finestra di esecuzione in un file formato PDF.

```
import processing.opengl.*;
import processing.pdf.*;
float sinLUT[];
float cosLUT[];
float SINCOS_PRECISION=1.0;
int SINCOS_LENGTH= int((360.0/SINCOS_PRECISION));
boolean dosave=false;
int num;
float pt[];
int style[];
void setup() {
  size(320,180, OPENGL);
  frameRate(24);
  background(255);
  sinLUT=new float[SINCOS_LENGTH];
  cosLUT=new float[SINCOS_LENGTH];
  for (int i = 0; i < SINCOS_LENGTH; i++) {
    sinLUT[i]= (float)Math.sin(i*DEG_TO_RAD*SINCOS_PRECISION);
    cosLUT[i]= (float)Math.cos(i*DEG_TO_RAD*SINCOS_PRECISION);
  }
  num = 150;
  pt = new float[6*num];
  style = new int[2*num];
  int index=0;
  float prob;
  for (int i=0; i<num; i++) {
    pt[index++] = random(PI*2);
```

```

    pt[index++] = random(PI*2);
    pt[index++] = random(60,80);
    if(random(100)>90) pt[index]=(int)random(8,27)*10;
    pt[index++] = int(random(2,50)*5);
    pt[index++] = random(4,32); // Width of band
    if(random(100)>90) pt[index]=random(40,60);
    pt[index++] = radians(random(5,30))/5;
    prob = random(100);
    if(prob<30) style[i*2]=colorBlended(random(1), 255,0,100, 255,0,0, 210);
    else if(prob<70) style[i*2]=colorBlended(random(1), 0,153,255, 170,225,255, 210);
    else if(prob<90) style[i*2]=colorBlended(random(1), 200,255,0, 150,255,0, 210);
    else style[i*2]=color(255,255,255, 220);
    if(prob<50) style[i*2]=colorBlended(random(1), 200,255,0, 50,120,0, 210);
    else if(prob<90) style[i*2]=colorBlended(random(1), 255,100,0, 255,255,0, 210);
    else style[i*2]=color(255,255,255, 220);
    style[i*2+1]=(int)(random(100))%3;
  }
}

void draw() {
  if(dosave) {
    PGraphicsPDF pdf = (PGraphicsPDF)beginRaw(PDF, "Complex3D.pdf");
    pdf.strokeJoin(MITER);
    pdf.strokeCap(SQUARE);
    pdf.fill(0);
    pdf.noStroke();
    pdf.rect(0,0, width,height);
  }
  background(0);
  int index=0;
  translate(width/2,height/2,0);
  rotateX(PI/6);
  rotateY(PI/6);
  for (int i=0; i<num; i++) {
    pushMatrix();
    rotateX(pt[index++]);
    rotateY(pt[index++]);
    if(style[i*2+1]==0) {
      stroke(style[i*2]);
      noFill();
      strokeWeight(1);
      arcLine(0,0, pt[index++],pt[index++],pt[index++]);
    }
    else if(style[i*2+1]==1) {
      fill(style[i*2]);
      noStroke();
      arcLineBars(0,0, pt[index++],pt[index++],pt[index++]);
    }
    else {
      fill(style[i*2]);
      noStroke();
      arc(0,0, pt[index++],pt[index++],pt[index++]);
    }
    pt[index-5]+=pt[index]/10;
    pt[index-4]+=pt[index++]/20;
  }
}

```

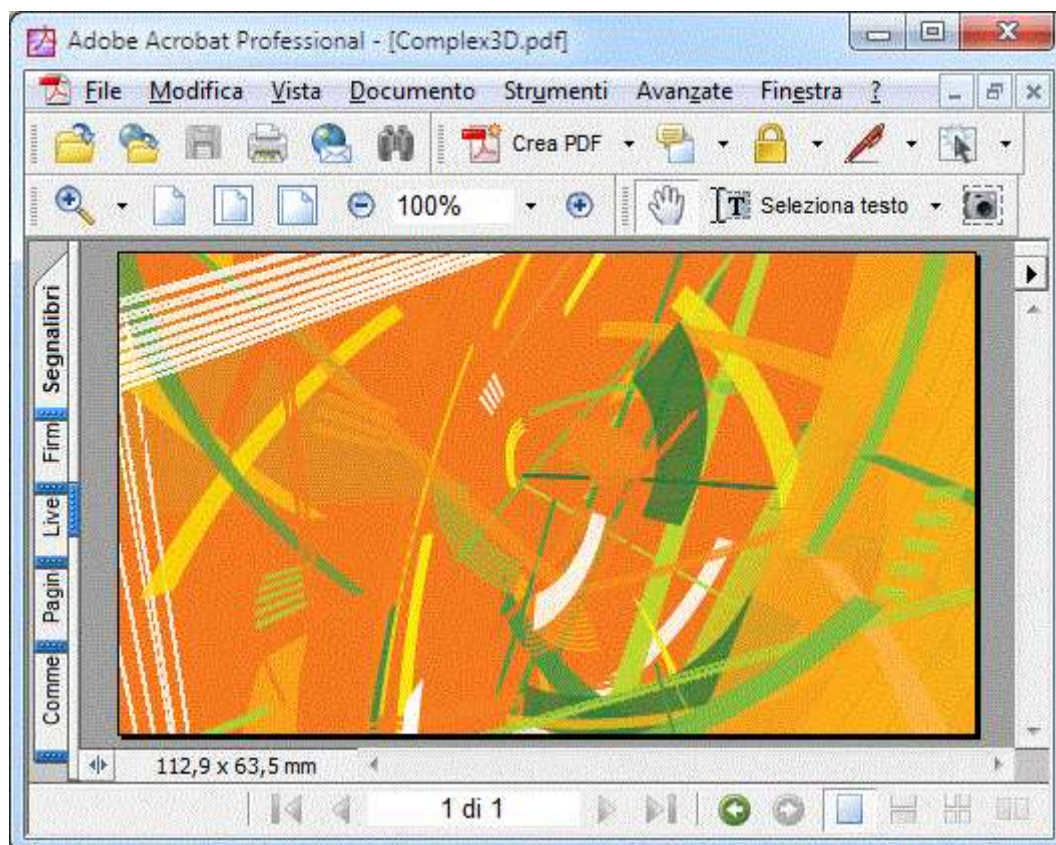
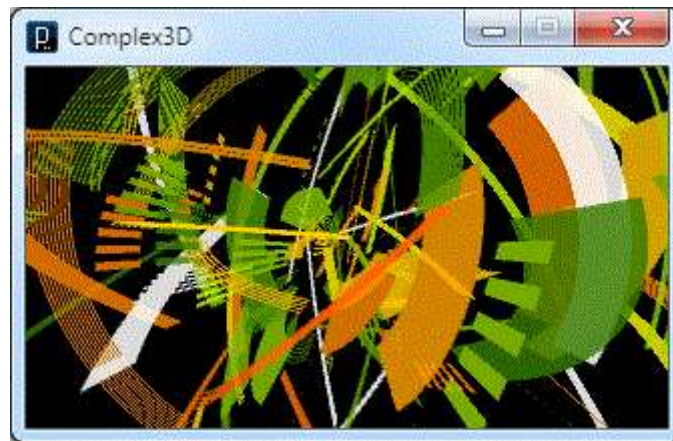
```

    popMatrix();
}
if(dosave) {
    endRaw();
    dosave=false;
}
}
public int colorBlended(float fract,
float r, float g, float b,
float r2, float g2, float b2, float a) {
    r2 = (r2 - r);
    g2 = (g2 - g);
    b2 = (b2 - b);
    return color(r + r2 * fract, g + g2 * fract, b + b2 * fract, a);
}
public void arcLine(float x,float y,float deg,float rad,float w) {
    int a=(int)(min (deg/SINCOS_PRECISION,SINCOS_LENGTH-1));
    int numlines=(int)(w/2);
    for (int j=0; j<numlines; j++) {
        beginShape();
        for (int i=0; i<a; i++) {
            vertex(cosLUT[i]*rad+x,sinLUT[i]*rad+y);
        }
        endShape();
        rad += 2;
    }
}
public void arcLineBars(float x,float y,float deg,float rad,float w) {
    int a = int((min (deg/SINCOS_PRECISION,SINCOS_LENGTH-1)));
    a /= 4;
    beginShape(QUADS);
    for (int i=0; i<a; i+=4) {
        vertex(cosLUT[i]*(rad)+x,sinLUT[i]*(rad)+y);
        vertex(cosLUT[i]*(rad+w)+x,sinLUT[i]*(rad+w)+y);
        vertex(cosLUT[i+2]*(rad+w)+x,sinLUT[i+2]*(rad+w)+y);
        vertex(cosLUT[i+2]*(rad)+x,sinLUT[i+2]*(rad)+y);
    }
    endShape();
}
public void arc(float x,float y,float deg,float rad,float w) {
    int a = int(min (deg/SINCOS_PRECISION,SINCOS_LENGTH-1));
    beginShape(QUAD_STRIP);
    for (int i = 0; i < a; i++) {
        vertex(cosLUT[i]*(rad)+x,sinLUT[i]*(rad)+y);
        vertex(cosLUT[i]*(rad+w)+x,sinLUT[i]*(rad+w)+y);
    }
    endShape();
}
void keyPressed() {
    if (key == 's') {
        dosave=true;
    }
}
void mouseReleased() {

```



```
background(255);  
}
```

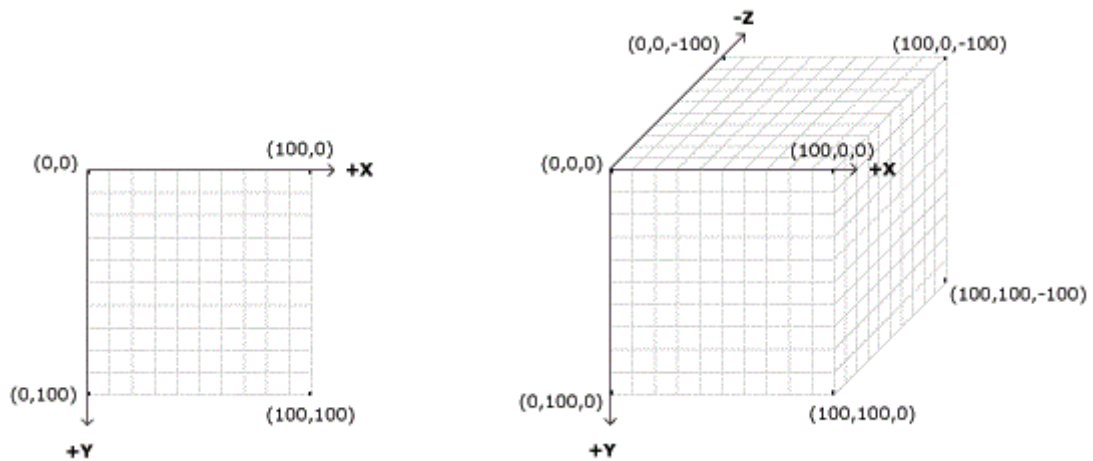


GRAFICA

SISTEMA DI COORDINATE

Processing usa il sistema di coordinate cartesiane con l'origine in alto a sinistra.

Se il programma apre una finestra di dimensioni larghezza 320 pixel e altezza 240 pixel, la coordinata [0,0] si trova in alto a sinistra, la coordinata [319,239] si trova il basso a destra.



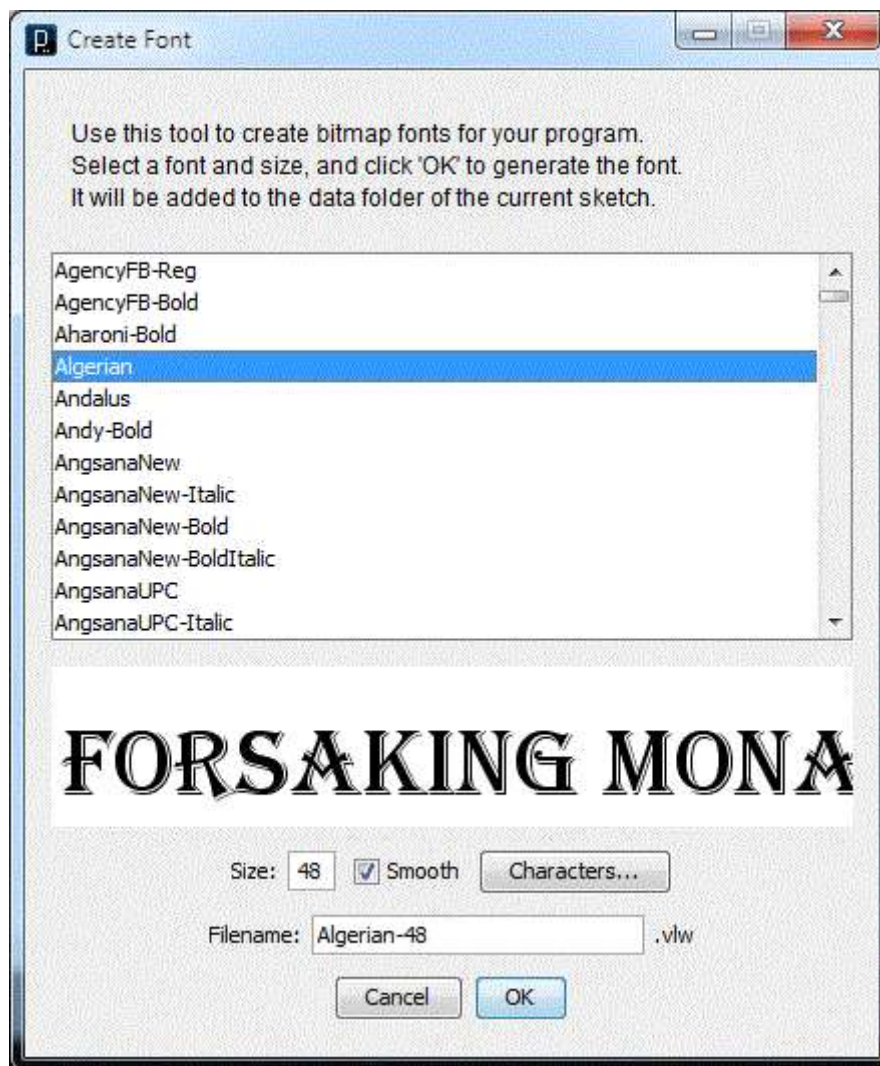
FONT E DISEGNO DI TESTI

In Processing è anche possibile disegnare testi; occorre creare il font dei caratteri e le relative dimensioni.

Per font s'intende un insieme di caratteri di aspetto omogeneo.

Quando si usa un font bisogna stabilire la dimensione dei caratteri, che si esprime in punti (pt) dove un punto è pari a 1/72 di pollice, in pratica circa 0.3527 mm, perché 1 inch è uguale a 2.54 cm.

Processing obbliga il programmatore a creare, prima della scrittura ed esecuzione del programma, i font che userà, tramite il menu **Tools/Create Font...**



La scelta dell'opzione apre una finestra che elenca i font installati e permette di creare una copia del font scelto con estensione VLW che è posto nella cartella DATA.

Poiché il programma usa la copia del font appena creato, non ci sono problemi di portabilità dei font.

Una volta creato un font, il programma può utilizzarlo; occorre definire una variabile di tipo *PFont* cui è assegnato con l'istruzione *loadFont* uno dei font creati che si trovano nella cartella DATA.

A questo punto, con il metodo *textFont* si sceglie il font caricato come font per il disegno di testi, fissando eventualmente anche la dimensione dei caratteri; in caso contrario, la dimensione è quella di default, assegnata al font al momento della sua creazione.

La dimensione di un font può essere variata con l'istruzione *textSize*.

Ogni lettera del testo è trattata come una figura piana chiusa, che può essere eventualmente colorata con il colore stabilito dal metodo *fill*.

Il colore di default è il bianco, anziché il nero come accade per le figure geometriche.

text(testo, x, y)

Disegna la stringa testo nello schermo posizionando il punto inferiore sinistro della stringa da disegnare alle coordinate (x, y).

loadFont(fontname)

Carica il font in una variabile di tipo *PFont*.

textFont(font)

textFont(font, size)

Definisce il font corrente che è stato in precedenza caricato con *loadFont*; il parametro *font* dev'essere una variabile di tipo *PFont*, il parametro *size* definisce la grandezza della lettera in pixel; se non è specificato *size*, il font corrente sarà impostato alla grandezza definita durante la creazione del font.

textSize(size)

Definisce la grandezza della lettera in pixel.

Esempio.

```
/* font_01
   disegno frasi con differenti font */
void setup()
{
  size (300,300);
  elab_font();
}
void elab_font()
{
  PFont font1;           // dichiarazione oggetto font1 tipo PFont
  PFont font2;           // dichiarazione oggetto font2 tipo PFont
  font1 = loadFont("Ellis-48.vlw"); // caricamento font Ellis-48
  font2 = loadFont("Joan-48.vlw");  // caricamento font Joan-48
  textFont(font1, 24);    // scelta font1 dimensione carattere 24
  fill(255,255,0);        // riempimento colore giallo
  text("ciao a tutti!", 50, 50);
  fill(0);                // riempimento colore nero
  text("ciao a tutti!", 50, 100);
  fill(255,0,0);          // riempimento colore rosso
  textFont(font2);        // scelta font2
  text("ciao a tutti!", 50, 150);
  fill(0,0,255);          // riempimento colore blu
  textSize(36);           // dimensione carattere 36
  text("ciao a tutti!", 50, 200);
}
```




ATTIBUTI GRAFICI

size(larghezza, altezza)

Imposta le dimensioni in pixel della finestra corrente; se non è impostata, la grandezza della finestra di default è 100X100 pixel.

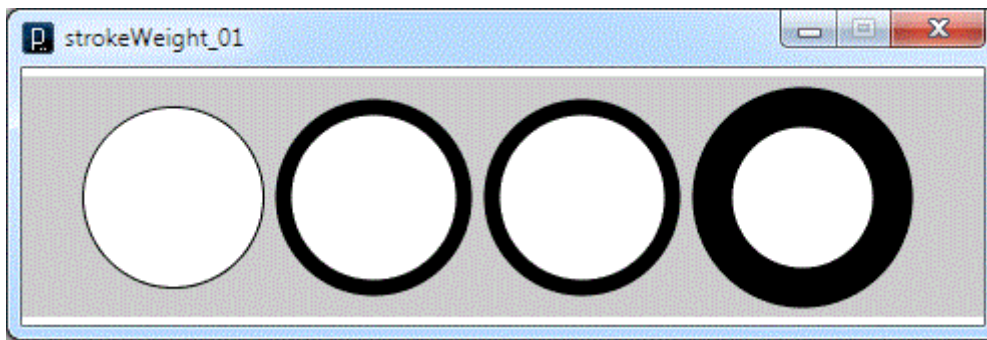
Una volta definita la dimensione della finestra, sono automaticamente impostate le variabili di sistema *width* e *height* che rappresentano rispettivamente la larghezza e l'altezza della finestra.

strokeWeight(spessore)

Imposta lo spessore della linea in pixel; di default Processing disegna la linea ad un pixel di spessore.

Esempio.

```
/* strokeWeight_01
   disegno ellissi con differenti spessori di linea */
size(480, 120);
smooth();                                // arrotonda i bordi
ellipse(75, 60, 90, 90);
strokeWeight(8);                          // spessore linea 8 pixel
ellipse(175, 60, 90, 90);
ellipse(279, 60, 90, 90);
strokeWeight(20);                         // spessore linea 20 pixel
ellipse(389, 60, 90, 90);
```



smooth

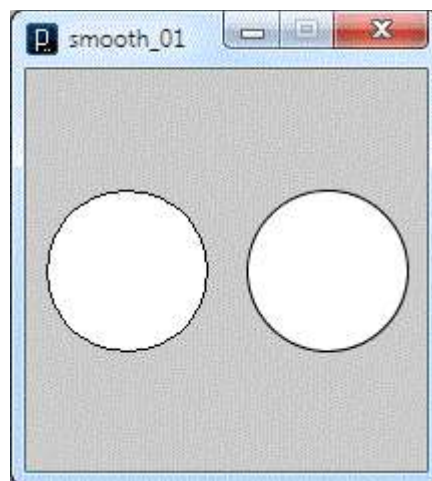
Arrotonda i bordi delle figure geometriche.

noSmooth

Disabilita la funzione *smooth*.

Esempio.

```
/* smooth_01
   Disegna un cerchio con i bordi arrotondati e
   un cerchio con i bordi non arrotondati */
void setup() {
  size(200, 200);
  smooth();
  ellipse(150, 100, 80, 80);
  noSmooth();
  ellipse(50, 100, 80, 80);
}
```



strokeJoin

strokeJoin(MITER)

strokeJoin(BEVEL)

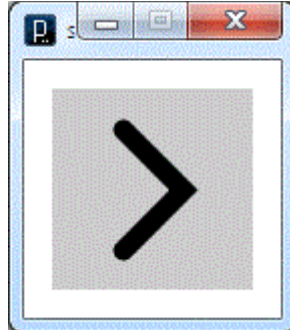
strokeJoin(ROUND)

Imposta lo stile nei punti di giunzione dei segmenti.

Esempio.

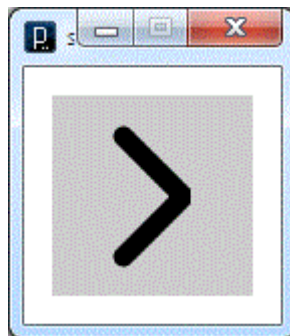
```
/* strokeJoin_01
   imposta lo stile nei punti di giunzione dei segmenti */
noFill();
```

```
smooth();
strokeWeight(10.0);
strokeJoin(MITER);
beginShape();
vertex(35, 20);
vertex(65, 50);
vertex(35, 80);
endShape();
```



Esempio.

```
/* strokeJoin_02
   imposta lo stile nei punti di giunzione dei segmenti */
noFill();
smooth();
strokeWeight(10);
strokeJoin(BEVEL);
beginShape();
vertex(35, 20);
vertex(65, 50);
vertex(35, 80);
endShape();
```

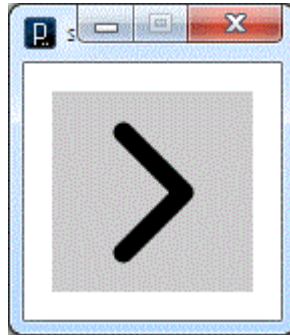


Esempio.

```
/* strokeJoin_03
   imposta lo stile nei punti di giunzione dei segmenti */
noFill();
smooth();
strokeWeight(10);
strokeJoin(ROUND);
beginShape();
vertex(35, 20);
vertex(65, 50);
```



```
vertex(35, 80);  
endShape();
```



COLORE

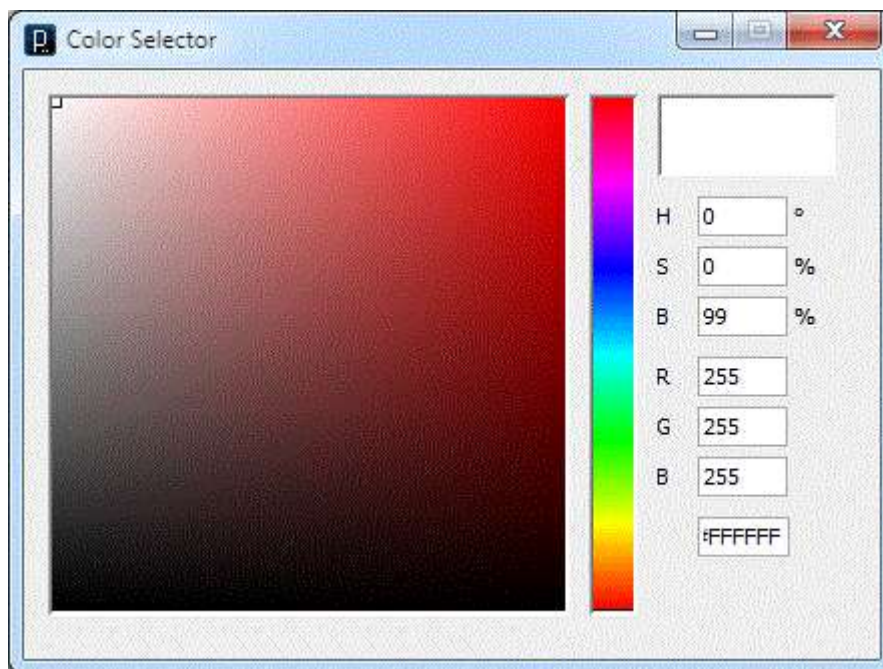
Il metodo *background* imposta il colore dello sfondo.

I metodi *stroke* e *fill* stabiliscono rispettivamente il colore del disegno dei contorni e l'area interna di una figura.

I tre numeri interi che costituiscono gli argomenti di questi metodi rappresentano un colore in formato **RGB** (*Red Green Blue*).

Al contrario, i metodi *noStroke* e *noFill* inibiscono rispettivamente il disegno del contorno e dell'interno di una figura, fino all'esecuzione di un nuovo comando *stroke* o *fill*.

Per determinare il codice RGB di un colore o per determinare il colore di una tripla RGB di numeri interi, si può usare il menu **Tools/Color Selector**.



È accettato anche un colore espresso in esadecimale.

Ad esempio, #FF00FF rappresenta il codice del colore magenta.

Il simbolo (#) specifica che i numeri seguenti sono cifre esadecimali.

Il riempimento effettuato con il seguente esempio è analogo.

```
fill(255,0,255);  
// oppure  
fill(#ff00ff);
```

Per il disegno di testi il colore di default è il bianco, per le figure geometriche il colore di

default è il nero.

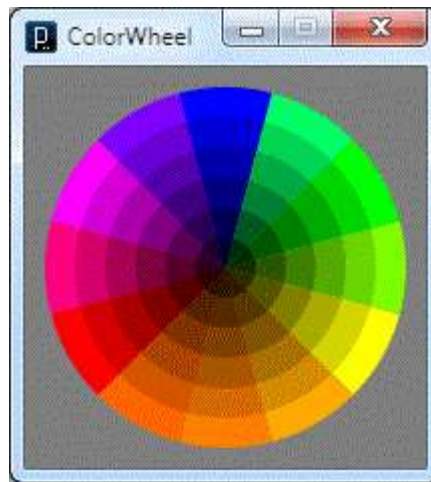
Esempio.

```
int segs = 12;
int steps = 6;
float rotAdjust = TWO_PI / segs / 2;
float radius;
float segWidth;
float interval = TWO_PI / segs;
void setup() {
  size(200, 200);
  background(127);
  smooth();
  ellipseMode(RADIUS);
  noStroke();
  // make the diameter 90% of the sketch area
  radius = min(width, height) * 0.45;
  segWidth = radius / steps;
  drawShadeWheel();
}
void drawShadeWheel() {
  for (int j = 0; j < steps; j++) {
    color[] cols = {
      color(255-(255/steps)*j, 255-(255/steps)*j, 0),
      color(255-(255/steps)*j, (255/1.5)-((255/1.5)/steps)*j, 0),
      color(255-(255/steps)*j, (255/2)-((255/2)/steps)*j, 0),
      color(255-(255/steps)*j, (255/2.5)-((255/2.5)/steps)*j, 0),
      color(255-(255/steps)*j, 0, 0),
      color(255-(255/steps)*j, 0, (255/2)-((255/2)/steps)*j),
      color(255-(255/steps)*j, 0, 255-(255/steps)*j),
      color((255/2)-((255/2)/steps)*j, 0, 255-(255/steps)*j),
      color(0, 0, 255-(255/steps)*j),
      color(0, 255-(255/steps)*j, (255/2.5)-((255/2.5)/steps)*j),
      color(0, 255-(255/steps)*j, 0),
      color((255/2)-((255/2)/steps)*j, 255-(255/steps)*j, 0)
    };
    for (int i = 0; i < segs; i++) {
      fill(cols[i]);
      arc(width/2, height/2, radius, radius,
          interval*i+rotAdjust, interval*(i+1)+rotAdjust);
    }
    radius -= segWidth;
  }
}
void drawTintWheel() {
  for (int j = 0; j < steps; j++) {
    color[] cols = {
      color((255/steps)*j, (255/steps)*j, 0),
      color((255/steps)*j, ((255/1.5)/steps)*j, 0),
      color((255/steps)*j, ((255/2)/steps)*j, 0),
      color((255/steps)*j, ((255/2.5)/steps)*j, 0),
      color((255/steps)*j, 0, 0),
      color((255/steps)*j, 0, ((255/2)/steps)*j),
    };
```

```

    color((255/steps)*j, 0, (255/steps)*j),
    color(((255/2)/steps)*j, 0, (255/steps)*j),
    color(0, 0, (255/steps)*j),
    color(0, (255/steps)*j, ((255/2.5)/steps)*j),
    color(0, (255/steps)*j, 0),
    color(((255/2)/steps)*j, (255/steps)*j, 0)
  };
  for (int i = 0; i < segs; i++) {
    fill(cols[i]);
    arc(width/2, height/2, radius, radius,
        interval*i+rotAdjust, interval*(i+1)+rotAdjust);
  }
  radius -= segWidth;
}
}
}

```



background

Imposta il colore dello sfondo.

background(gray)

gray imposta i livelli di grigio, scelti tra 256 tonalità diverse: zero per il nero e 255 per il bianco.

background (R,G,B)

Colore impostato in formato RGB.

background (hex)

Colore impostato in formato esadecimale.

stroke

Imposta il colore del contorno.

stroke(gray)

gray imposta i livelli di grigio, scelti tra 256 tonalità diverse: zero per il nero e 255 per il bianco.

stroke(R,G,B)

Colore impostato in formato RGB.

stroke(hex)

Colore impostato in formato esadecimale.

noStroke

Inibisce il disegno del contorno fino all'esecuzione di un nuovo comando *stroke*.

fill

Imposta il colore di riempimento.

fill(gray)

gray imposta i livelli di grigio, scelti tra 256 tonalità diverse: zero per il nero e 255 per il bianco.

fill (R,G,B)

Colore impostato in formato RGB.

fill (hex)

Colore impostato in formato esadecimale.

noFill

Disabilita il riempimento.

Esempio.

<i>background(0);</i>	<i>// background colore nero</i>
<i>background(0,255,0);</i>	<i>// background colore verde</i>
<i>stroke(204, 102,0);</i>	<i>// contorno colore arancione</i>
<i>stroke(180,0,0);</i>	<i>// contorno colore rosso</i>
<i>stroke(0,180,0);</i>	<i>// contorno colore verde</i>
<i>stroke(0,0,180);</i>	<i>// contorno colore blu</i>
<i>fill(0,255,0);</i>	<i>// riempimento colore verde</i>
<i>fill(255,255,0);</i>	<i>// riempimento colore giallo</i>
<i>fill(150,250,250);</i>	<i>// riempimento colore azzurro</i>
<i>fill(230,150,250);</i>	<i>// riempimento colore viola</i>
<i>fill(240,220,220);</i>	<i>// riempimento colore rosa</i>
<i>fill(0,180,0);</i>	<i>// riempimento colore verde</i>
<i>fill(180,0,0);</i>	<i>// riempimento colore rosso</i>
<i>fill(153);</i>	<i>// riempimento colore grigio</i>
<i>fill(0,0,153);</i>	<i>// riempimento colore blu</i>

METODI

In Processing si possono disegnare punti, linee, superfici e volumi ovvero oggetti geometrici a una, due o tre dimensioni.

Il numero di parametri delle primitive corrispondenti determinerà se tali oggetti debbano essere collocati nello spazio (X, Y) o nello spazio (X, Y, Z).

A ogni oggetto si possono modificare gli attributi; può essere eventualmente colorato con il colore stabilito dal metodo *fill*; il colore di default per le figure geometriche è il nero.

Altri attributi possono essere il colore e lo spessore del bordo e smussature degli angoli.

point(x,y)

Disegna un punto alle coordinate (x, y).

Esempio.

```
/* point_01  
disegna un punto alle coordinate (240,60) */  
size(480, 120);  
point(240, 60);
```

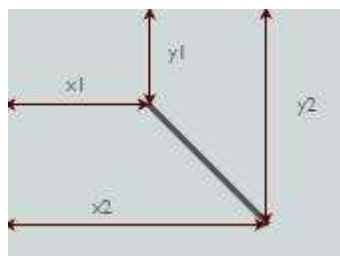


```
/* point_02  
disegna un punto alle coordinate (60,60), colore rosso, dimensione  
pixel 10 */  
size(480, 120);  
stroke(180,0,0);           // colore rosso  
strokeWeight(10);          // dimensione 10 pixel  
point(60,60);
```



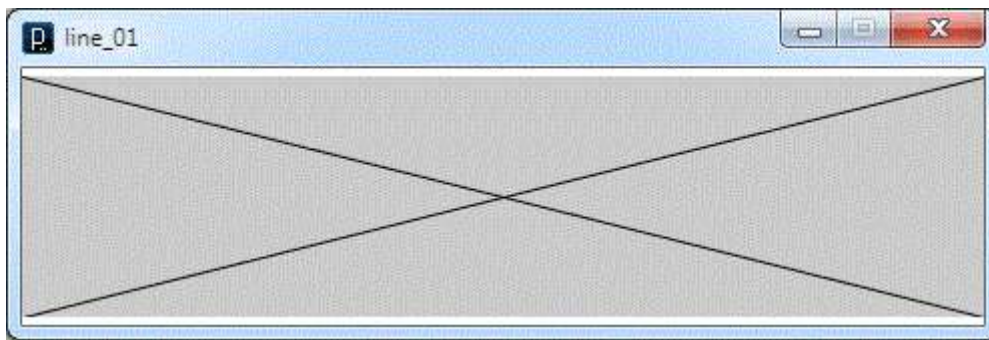
line(x1, y1, x2, y2)

Traccia un segmento di retta tra due punti di coordinate (x1, y1) e (x2, y2).



Esempio.

```
/* line_01  
disegna due linee corrispondenti alle diagonali dello schermo */  
size(480, 120);  
line(0, 0, width, height);           // Line da (0,0) a (480, 120)  
line(width, 0, 0, height);           // Line da (480, 0) a (0, 120)
```

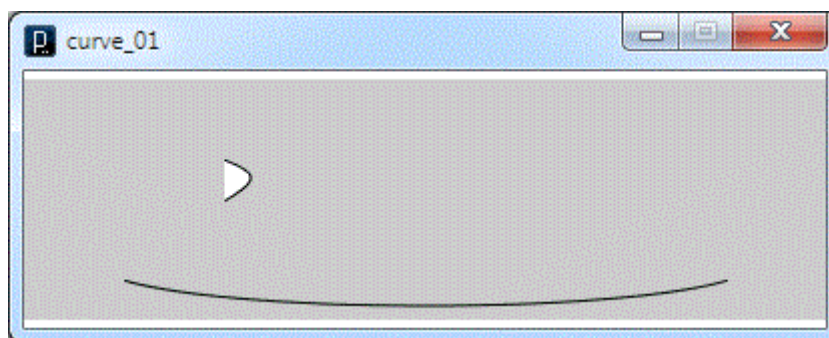


curve(x1, y1, x2, y2, x3, y3, x4, y4)

Invocata con otto parametri, traccia una curva, con punto iniziale e finale determinati, dalla seconda e dalla terza coppia di coordinate passate come argomenti; le due coppie di coordinate iniziale e finale definiscono due punti di controllo della curva tracciata; è pertanto visualizzato solo il segmento di curva compreso tra i due punti intermedi.

Esempio.

```
/* curve_01
   disegna due curve */
size(400,120);
curve(0, 20, 100, 40, 100, 60, 0, 100);
noFill(); // disabilita riempimento
curve(0, 0, 50, 100, 350, 100, 400, 0);
```



curveVertex(x,y)

Una curva generica è definita da una successione di vertici ed è un oggetto dotato di una superficie che può essere colorata.

In Processing tali vertici, definiti dalla funzione `curveVertex(x,y)`, si elencano tra `beginShape` ed `endShape`; le due coppie di coordinate iniziale e finale definiscono due punti di controllo della curva tracciata; è pertanto visualizzato solo il segmento di curva compreso tra i punti intermedi.

Esempio.

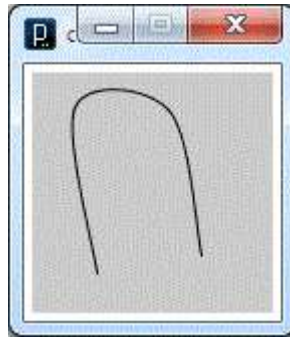
```
/* curveVertex_01
   disegna una curva passante tra i punti A, B, C, D; i punti iniziale
   e finale definiscono due punti di controllo della curva tracciata */
size(120,120);
noFill(); // disabilita riempimento
beginShape();
curveVertex(84, 91); // punto iniziale
curveVertex(84, 91); // punto A
curveVertex(68, 19); // punto B
```



```

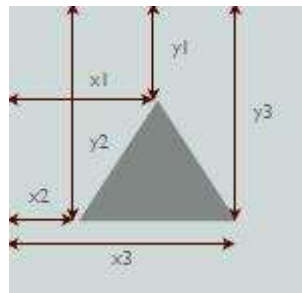
curveVertex(21, 17);           // punto C
curveVertex(32, 100);          // punto D
curveVertex(32, 100);          // punto finale
endShape();

```



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

Invocata con sei parametri corrispondenti alle coordinate dei tre vertici del triangolo.

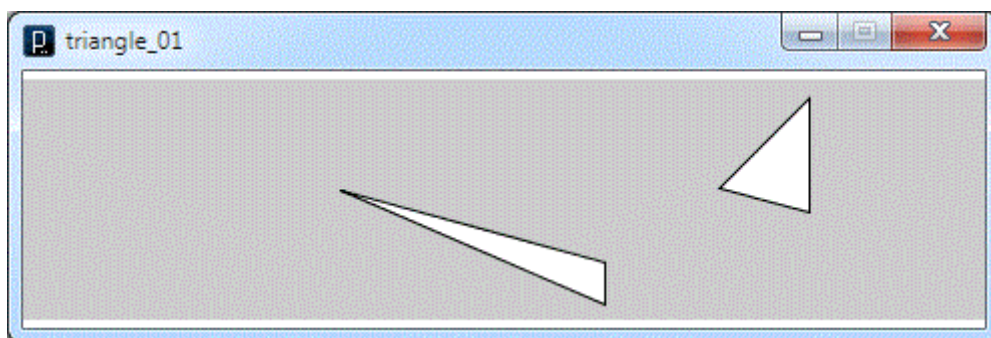


Esempio.

```

/* triangle_01
   disegna due triangoli */
size(480, 120);
triangle(158, 55, 290, 91, 290, 112);
triangle(347, 54, 392, 9, 392, 66);

```

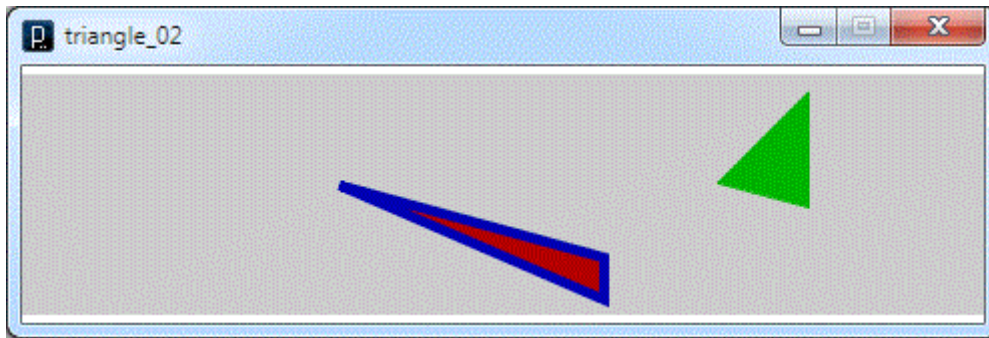


```

/* triangle_02
   disegna due triangoli, definendo contorno e riempimento */
size(480, 120);
stroke(0,180,0);           // contorno colore verde
fill(0,180,0);              // riempimento colore verde
triangle(347, 54, 392, 9, 392, 66);
stroke(0,0,180);            // contorno colore blu
strokeWeight(5);             // spessore linea 5 pixel
fill(180,0,0);              // riempimento colore rosso

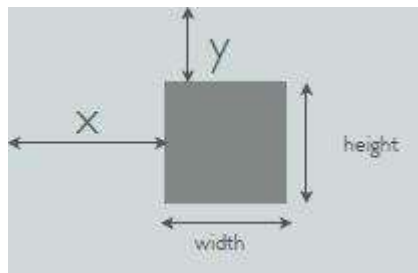
```

```
triangle(158, 55, 290, 91, 290, 112);
```



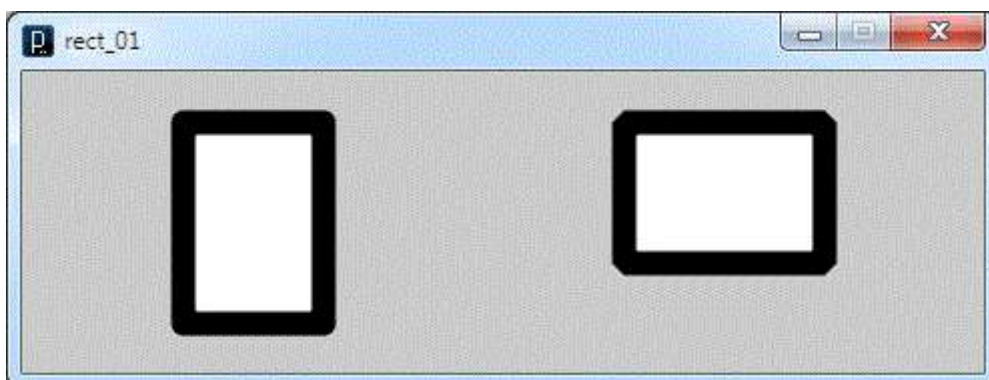
rect(x, y, width, height)

Invocata con quattro parametri, in cui i primi due specificano la posizione dell'angolo in alto a sinistra, il terzo e quarto specificano, rispettivamente, la larghezza e l'altezza, disegna un rettangolo; se *width* = *height* disegna un quadrato.



Esempio.

```
/* rect_01
   disegna due rettangoli arrotondando gli angoli */
size(480, 150);
smooth();                                // arrotonda i bordi
strokeWeight(12);                         // spessore linea 12 pixel
strokeJoin(ROUND);                        // arrotonda gli angoli
rect(80, 25, 70, 100);
strokeJoin(BEVEL);                         // smussa gli angoli
rect(300, 25, 100, 70);
```

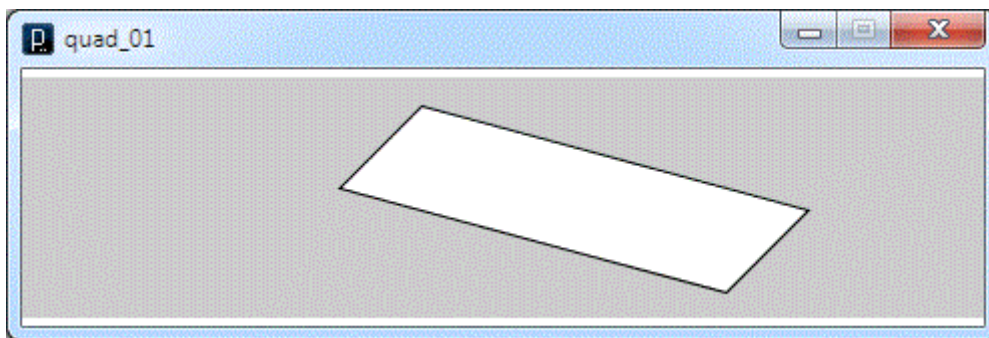


quad(x1, y1, x2, y2, x3, y3, x4, y4)

Disegna un quadrilatero generico mediante le coordinate dei suoi quattro vertici, passate come parametri alla funzione *quad*.

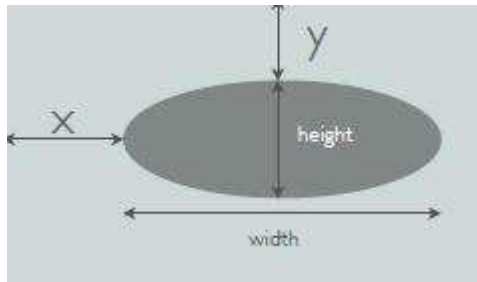
Esempio.

```
/* quad_01  
   disegna un quadrilatero */  
size(480, 120);  
quad(158, 55, 199, 14, 392, 66, 351, 107);
```



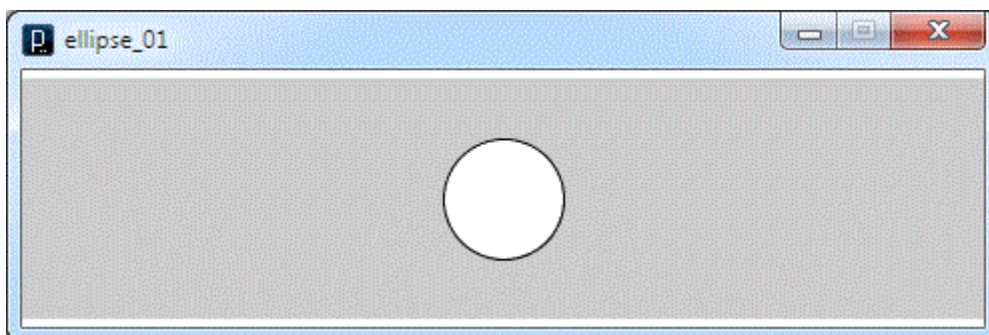
ellipse(x, y, width, height)

Disegna un ellisse di centro (x, y), diametro orizzontale *width*, diametro verticale *height*; se i due diametri sono uguali, disegna una circonferenza.



Esempio.

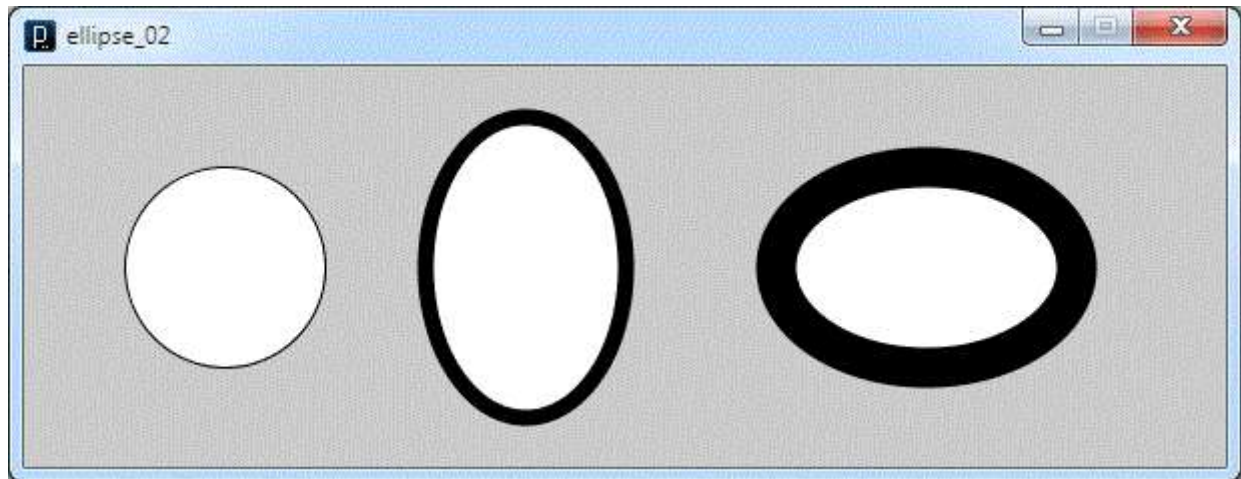
```
/* ellipse_01  
   disegna un cerchio posizionato al centro dello schermo,raggio=60 */  
size(480, 120);  
ellipse(width/2, height/2, 60, 60);
```



Esempio.

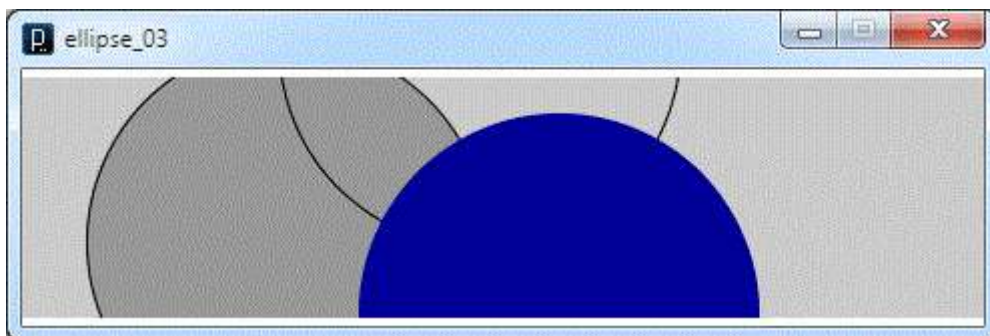
```
/* ellipse_02  
   disegna un cerchio e due ellissi */  
size(600, 200);  
smooth();  
ellipse(100, 100, 100, 100);
```

```
strokeWeight(8);           // spessore linea 8 pixels
ellipse(250, 100, 100, 150);
strokeWeight(20);          // spessore linea 20 pixels
ellipse(450, 100, 150, 100);
```



Esempio.

```
/* ellipse_03
   disegna tre ellissi, definendo colore riempimento, bordo,
   nessun bordo */
size(480, 120);
smooth();           // arrotonda il bordo
fill(153);          // riempimento colore grigio
ellipse(132, 82, 200, 200);
noFill();           // disabilita riempimento
ellipse(228, -16, 200, 200);
noStroke();         // nessun bordo
fill(0,0,153);      // riempimento colore blu
ellipse(268, 118, 200, 200);
```



arc(x, y, width, height, start, stop)

Disegna un arco di centro (x, y), diametro orizzontale *width*, diametro verticale *height*, angolo inizio dell'arco in radianti, angolo fine dell'arco in radianti; se i due diametri sono uguali, disegna una porzione di circonferenza.

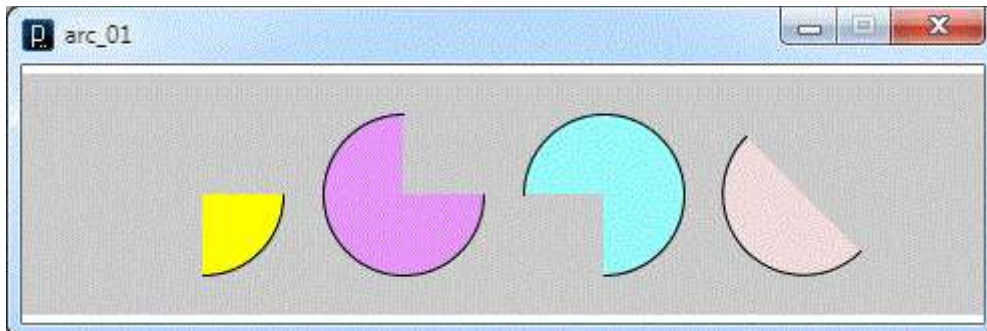
Esempio.

```
/* arc_01
   disegna quattro archi, definendo colore riempimento */
size(480, 120);
```

```

fill(255,255,0);           // riempimento colore giallo
arc(90, 60, 80, 80, 0, HALF_PI);
fill(230,150,250);         // riempimento colore viola
arc(190, 60, 80, 80, 0, PI+HALF_PI);
fill(150,250,250);         // riempimento colore azzurro
arc(290, 60, 80, 80, PI, TWO_PI+HALF_PI);
fill(240,220,220);         // riempimento colore rosa
arc(390, 60, 80, 80, QUARTER_PI, PI+QUARTER_PI);

```



Poligoni

Un poligono generico è definito da una successione di vertici ed è un oggetto dotato di una superficie che può essere colorata.

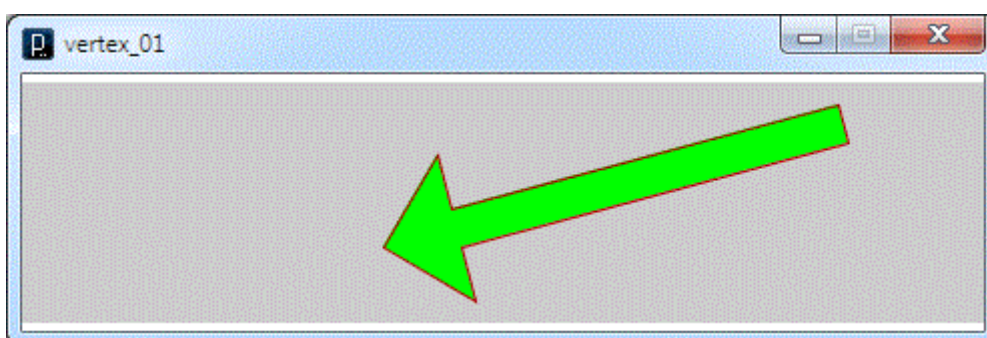
In Processing tali vertici, definiti dalla funzione `vertex(x,y)`, si elencano tra `beginShape` ed `endShape(CLOSE)`.

Esempio.

```

/* vertex_01
   disegna un poligono a forma di freccia, definendo bordo e
   colore riempimento */
size(480, 120);
stroke(180,0,0);           // bordo colore rosso
fill(0,255,0);             // riempimento colore verde
beginShape();
vertex(180, 82);
vertex(207, 36);
vertex(214, 63);
vertex(407, 11);
vertex(412, 30);
vertex(219, 82);
vertex(226, 109);
endShape(CLOSE);

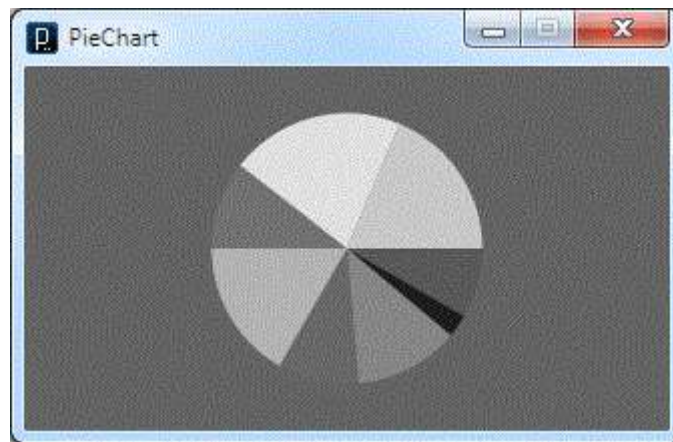
```



Grafici a torta

Esempio.

```
size(320, 180);
background(100);
smooth();
noStroke();
float diameter = min(width, height) * 0.75;
int[] ang = {30, 10, 45, 35, 60, 38, 75, 67};
float lastAng = 0;
for (int i = 0; i < ang.length; i++){
  fill(ang[i] * 3.0);
  arc(width/2, height/2, diameter, diameter, lastAng, lastAng+radians(ang[i]));
  lastAng += radians(ang[i]);
}
```



CARICAMENTO IMMAGINE

Processing ha il tipo di dato *PImage*.

Un oggetto di tipo *PImage* rappresenta un'immagine, caricata da un file di formato **JPEG** (*Joint Photographic Experts Group*), **PNG** (*Portable Network Graphics*) oppure **GIF** (*Graphics Interchange Format*).

Le immagini devono trovarsi nella cartella DATA contenuta nella cartella del programma che le manipola.

Gli oggetti di tipo *PImage* sono descritti da tre campi o attributi, che rappresentano rispettivamente la larghezza, l'altezza e la sequenza dei pixel dell'immagine.

Per accedere agli attributi di un'immagine si scrive il nome dell'immagine seguito da un attributo: ad esempio *d.width* rappresenta la larghezza dell'immagine *d*, mentre *d.height* rappresenta l'altezza dell'immagine *d*.

Tali valori sono usati affinché la finestra di disegno comprenda esattamente le immagini.

Per caricare un'immagine nel programma è utilizzata la funzione *loadImage*.

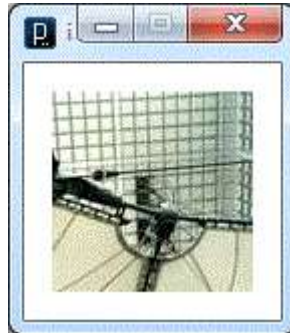
```
PImage immagine;
immagine=loadImage("foto.jpg");
```

Per disegnare un'immagine sullo schermo, posizionandone l'angolo superiore sinistro alle coordinate (x,y), è utilizzata la funzione *image*.

```
image(immagine,x,y);
```

Esempio.

```
/* image_01
   carica un'immagine */
PImage b;
b = loadImage("immagine_01.JPG");
image(b, 0, 0);
```



ANIMAZIONI

Omettendo il metodo *draw* oppure inserendo nel metodo *setup* il metodo *noloop* che impedisce di eseguire più di una volta il metodo *draw*, non è possibile realizzare delle animazioni.

Pertanto, per realizzare delle animazioni grafiche occorre inserire il metodo *draw* con le opportune istruzioni grafiche.

Esempio, semplice animazione.

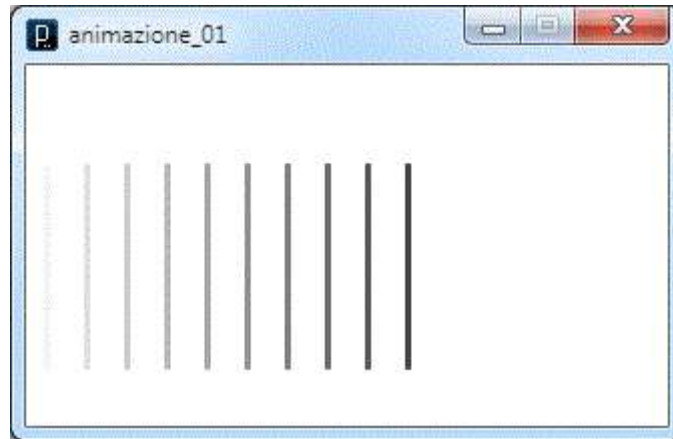
```
/* animazione_01
   sono disegnate delle linee con una tonalità di grigio
   che risulterà più scura per valori crescenti della
   coordinata x; se la finestra è completamente occupata,
   la si cancella e si ricomincia da capo */
int x;
void setup() {
  size(320, 180);
  strokeWeight(3);
  background(255);
  smooth();
  x = 10;
}
void draw() {
  if(x < 250){
    disegnaLinea(x, 50, x, 150);
    x = x + 20;
  }
  else {
    x = 10;
    background(255);
  };
  frameRate(2);
}
void disegnaLinea(int x1, int y1, int x2, int y2){
  stroke(255 - x1);
```



```

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

```



Animazioni d'immagini

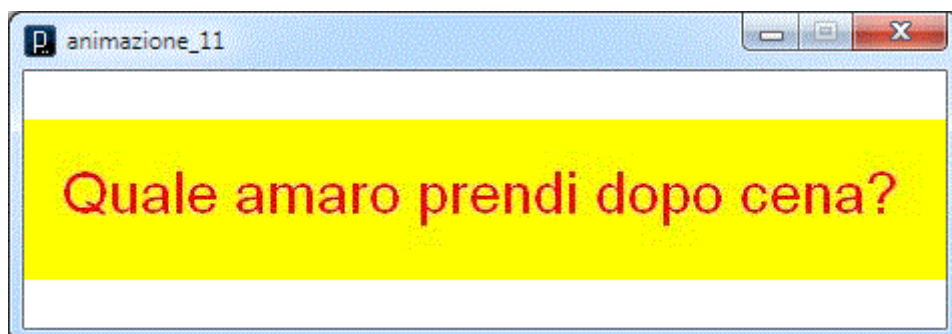
Un altro modo di produrre un'animazione è di visualizzare velocemente sullo schermo una sequenza d'immagini.

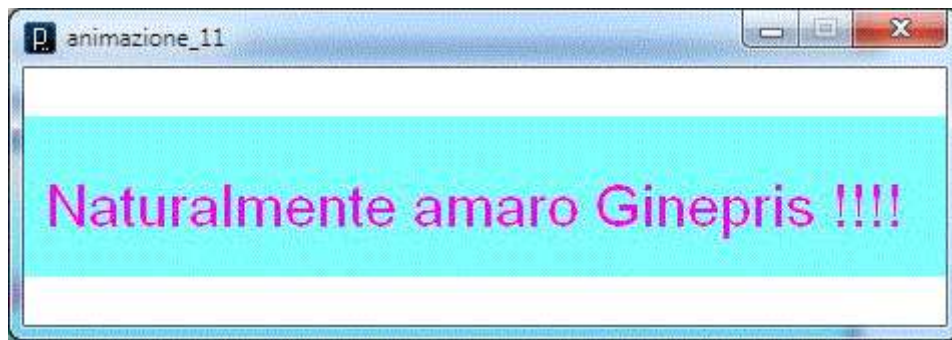
Esempio.

```

/* animazione_11
   realizza un banner pubblicitario, usando due immagini che si
   alternano sullo schermo ad intervalli regolari di un secondo */
PImage d, r;
void setup() {
  d = loadImage("domanda.JPG");
  r = loadImage("risposta.JPG");
  size(d.width, d.height);
  frameRate(1);
}
void draw() {
  if (frameCount%2==1)
    image(d, 0, 0);
  else
    image(r, 0, 0);
}

```





frameRate

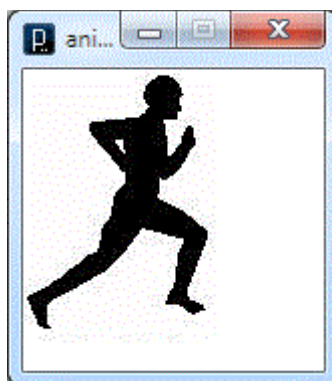
Permette di visualizzare n fotogrammi (frame) in un secondo; il valore di default è 60.

Esempio, animazione costituita da sei fotogrammi che ritraggono il profilo di un uomo che corre; provare con 5, 10, 20 e 60 fotogrammi per secondo e provare a commentare l'istruzione *frameRate*.

Il metodo *setup* carica le sei immagini dell'animazione assegnandole agli oggetti *a[0]*, ... , *a[5]* di tipo *PImage*.

Il metodo *draw* disegna l'immagine *a[k]* dove *k* è il resto della divisione per sei della variabile *frameCount*.

```
/* animazione_12
   realizza un'animazione costituita da sei fotogrammi che
   ritraggono il profilo di un uomo che corre */
PImage [] a = new PImage[6];
void setup() {
  size(150, 150);
  for (int i=0; i<6; i++)
    a[i] = loadImage("corridore"+i+".JPG");
  frameRate(10);
}
void draw() {
  image(a[frameCount%6], 0, 0);
}
```



Animazioni testuali

Un altro modo di produrre animazioni simili a quella del banner pubblicitario è quello di visualizzare in fotogrammi diversi delle frasi.

La struttura del programma è simile a quella dei programmi precedenti.

Invece di disegnare delle figure geometriche o di visualizzare delle immagini usando il metodo *image*, si disegnano delle stringhe di testo usando il metodo *text*.

delay

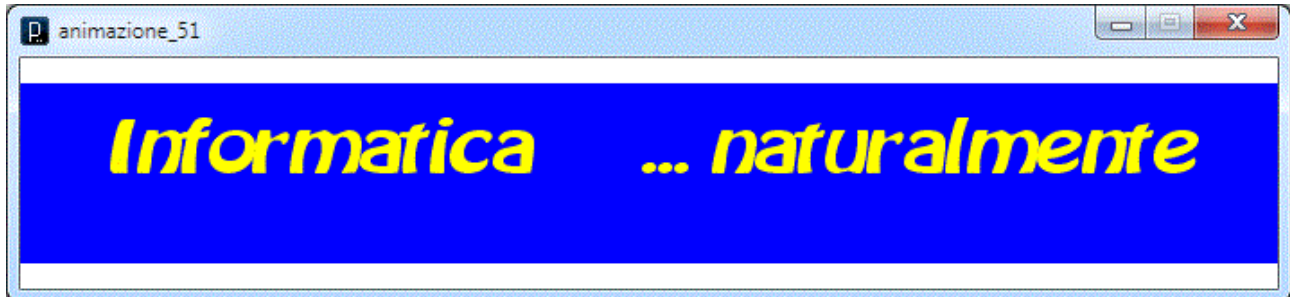
Regola la temporizzazione della sequenza d'immagini da visualizzare, analogamente a *frameRate*, con la differenza che *delay* ferma l'esecuzione del programma per un tempo espresso in millisecondi pari al valore intero che gli è passato come parametro, nell'esempio 2000 msec = 2 sec.

Il metodo *delay* si usa quando il tempo di permanenza di un fotogramma sullo schermo è molto alto e non è possibile usare il metodo *frameRate*, perché esso stabilisce il numero di fotogrammi da visualizzare in un secondo.

Esempio, provare con temporizzazione 500 e 5000.

```
/* animazione_51
   realizza un banner pubblicitario, usando delle stringhe di
   testo che si alternano sullo schermo */
PFont font1;
int c=0;
String [] a = new String[3];
void setup(){
  size(700, 100);
  font1 = loadFont("Batavia-36.vlw");
  a[0]="sapete qual e'";
  a[1]="il migliore corso del Fauser?";
  a[2]="Informatica    ... naturalmente";
  textFont(font1);
  fill(255, 255, 0);          // riempimento colore giallo
  noStroke();
}
void draw() {
  background(0, 0, 255);
  text(a[c], 50, 50);
  if ( c < 2 )
    c++;
  else
    c=0;
  delay(2000);
}
```





GUI (*GRAPHICS USER INTERFACE*)

BUTTON

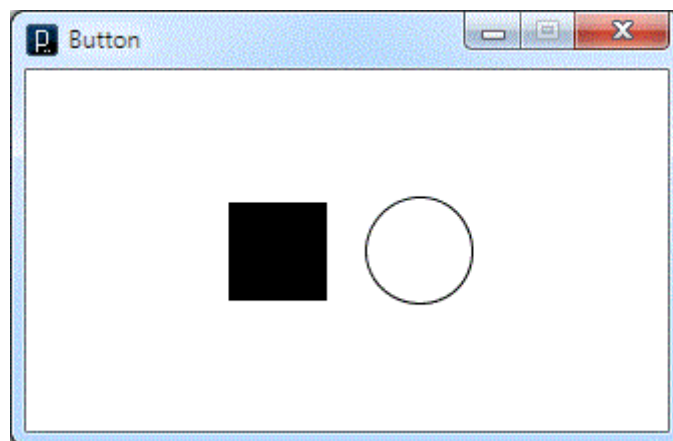
Esempio, fare clic al centro del quadrato per cambiare il colore di background.

```
int rectX, rectY;           // posizione del pulsante quadrato
int circleX, circleY;       // posizione del pulsante cerchio
int rectSize = 50;          // diametro del quadrato
int circleSize = 53;        // diametro del cerchio
color rectColor, circleColor, baseColor;
color rectHighlight, circleHighlight;
color currentColor;
boolean rectOver = false;
boolean circleOver = false;
void setup()
{
  size(320, 180);
  smooth();
  rectColor = color(0);
  rectHighlight = color(51);
  circleColor = color(255);
  circleHighlight = color(204);
  baseColor = color(102);
  currentColor = baseColor;
  circleX = width/2+circleSize/2+10;
  circleY = height/2;
  rectX = width/2-rectSize-10;
  rectY = height/2-rectSize/2;
  ellipseMode(CENTER);
}
void draw()
{
  update(mouseX, mouseY);
  background(currentColor);
  if(rectOver) {
    fill(rectHighlight);
  } else {
    fill(rectColor);
  }
  stroke(255);
  rect(rectX, rectY, rectSize, rectSize);
  if(circleOver) {
    fill(circleHighlight);
  } else {
    fill(circleColor);
  }
  stroke(0);
  ellipse(circleX, circleY, circleSize, circleSize);
}
void update(int x, int y)
```

```

{
  if( overCircle(circleX, circleY, circleSize) ) {
    circleOver = true;
    rectOver = false;
  } else if ( overRect(rectX, rectY, rectSize, rectSize) ) {
    rectOver = true;
    circleOver = false;
  } else {
    circleOver = rectOver = false;
  }
}
void mousePressed()
{
  if(circleOver) {
    currentColor = circleColor;
  }
  if(rectOver) {
    currentColor = rectColor;
  }
}
boolean overRect(int x, int y, int width, int height)
{
  if (mouseX >= x && mouseX <= x+width &&
    mouseY >= y && mouseY <= y+height) {
    return true;
  } else {
    return false;
  }
}
boolean overCircle(int x, int y, int diameter)
{
  float disX = x - mouseX;
  float disY = y - mouseY;
  if(sqrt(sq(disX) + sq(disY)) < diameter/2 ) {
    return true;
  } else {
    return false;
  }
}
}

```



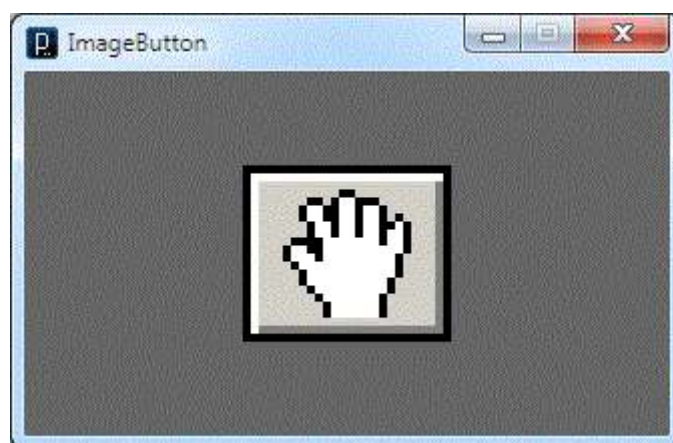
Esempio, disegnare un pulsante con immagine.

```
ImageButtons button;
void setup()
{
  size(320, 180);
  background(102, 102, 102);
  // definisce e crea il pulsante
  PImage b = loadImage("base.gif");
  PImage r = loadImage("roll.gif");
  PImage d = loadImage("down.gif");
  int x = width/2 - b.width/2;
  int y = height/2 - b.height/2;
  int w = b.width;
  int h = b.height;
  button = new ImageButtons(x, y, w, h, b, r, d);
}
void draw()
{
  button.update();
  button.display();
}
class Button
{
  int x, y;
  int w, h;
  color basecolor, highlightcolor;
  color currentcolor;
  boolean over = false;
  boolean pressed = false;
  void pressed() {
    if(over && mousePressed) {
      pressed = true;
    } else {
      pressed = false;
    }
  }
  boolean overRect(int x, int y, int width, int height) {
    if (mouseX >= x && mouseX <= x+width &&
        mouseY >= y && mouseY <= y+height) {
      return true;
    } else {
      return false;
    }
  }
}
class ImageButtons extends Button
{
  PImage base;
  PImage roll;
  PImage down;
  PImage currentimage;
  ImageButtons(int ix, int iy, int iw, int ih, PImage ibase, PImage iroll, PImage idown)
  {
```

```

x = ix;
y = iy;
w = iw;
h = ih;
base = ibase;
roll = iroll;
down = idown;
currentimage = base;
}
void update()
{
  over();
  pressed();
  if(pressed) {
    currentimage = down;
  } else if (over){
    currentimage = roll;
  } else {
    currentimage = base;
  }
}
void over()
{
  if( overRect(x, y, w, h) ) {
    over = true;
  } else {
    over = false;
  }
}
void display()
{
  image(currentimage, x, y);
}
}

```



Esempio, trascinare il quadratino bianco per cambiargli la posizione.

```

Handle[] handles;
int num;
void setup()
{

```

```

size(320, 180);
num = height/15;
handles = new Handle[num];
int hsize = 10;
for(int i=0; i<num; i++) {
    handles[i] = new Handle(width/2, 10+i*15, 50-hsize/2, 10, handles);
}
}
void draw()
{
    background(153);
    for(int i=0; i<num; i++) {
        handles[i].update();
        handles[i].display();
    }

    fill(0);
    rect(0, 0, width/2, height);
}
void mouseReleased()
{
    for(int i=0; i<num; i++) {
        handles[i].release();
    }
}
class Handle
{
    int x, y;
    int boxx, boxy;
    int length;
    int size;
    boolean over;
    boolean press;
    boolean locked = false;
    boolean otherslocked = false;
    Handle[] others;
    Handle(int ix, int iy, int il, int is, Handle[] o)
    {
        x = ix;
        y = iy;
        length = il;
        size = is;
        boxx = x+length - size/2;
        boxy = y - size/2;
        others = o;
    }
    void update()
    {
        boxx = x+length;
        boxy = y - size/2;
        for(int i=0; i<others.length; i++) {
            if(others[i].locked == true) {
                otherslocked = true;
                break;
            }
        }
    }
}

```

```

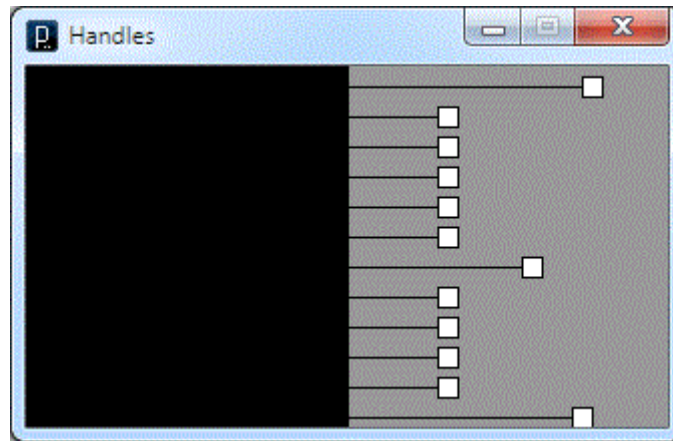
    } else {
        otherslocked = false;
    }
}
if(otherslocked == false) {
    over();
    press();
}
if(press) {
    length = lock(mouseX-width/2-size/2, 0, width/2-size-1);
}
}
void over()
{
    if(overRect(boxx, boxy, size, size)) {
        over = true;
    } else {
        over = false;
    }
}
void press()
{
    if(over && mousePressed || locked) {
        press = true;
        locked = true;
    } else {
        press = false;
    }
}
void release()
{
    locked = false;
}
void display()
{
    line(x, y, x+length, y);
    fill(255);
    stroke(0);
    rect(boxx, boxy, size, size);
    if(over || press) {
        line(boxx, boxy, boxx+size, boxy+size);
        line(boxx, boxy+size, boxx+size, boxy);
    }
}
}
boolean overRect(int x, int y, int width, int height)
{
    if (mouseX >= x && mouseX <= x+width &&
        mouseY >= y && mouseY <= y+height) {
        return true;
    } else {
        return false;
    }
}
}

```

```

int lock(int val, int minv, int maxv)
{
    return min(max(val, minv), maxv);
}

```



Esempio, disegnare una scrollbar, spostarla a sinistra e a destra per modificare la posizione dell'immagine.

```

HScrollbar hs1, hs2;
PImage top, bottom;
int topWidth, bottomWidth;
void setup()
{
    size(320, 180);
    noStroke();
    hs1 = new HScrollbar(0, 20, width, 10, 3*5+1);
    hs2 = new HScrollbar(0, height-20, width, 10, 3*5+1);
    top = loadImage("seedTop.jpg");
    topWidth = top.width;
    bottom = loadImage("seedBottom.jpg");
    bottomWidth = bottom.width;
}
void draw()
{
    background(255);
    float topPos = hs1.getPos()-width/2;
    fill(255);
    image(top, width/2-topWidth/2 + topPos*2, 0);
    float bottomPos = hs2.getPos()-width/2;
    fill(255);
    image(bottom, width/2-bottomWidth/2 + bottomPos*2, height/2);
    hs1.update();
    hs2.update();
    hs1.display();
    hs2.display();
}
class HScrollbar
{
    int swidth, sheight;
    int xpos, ypos;
    float spos, newspos;
}

```

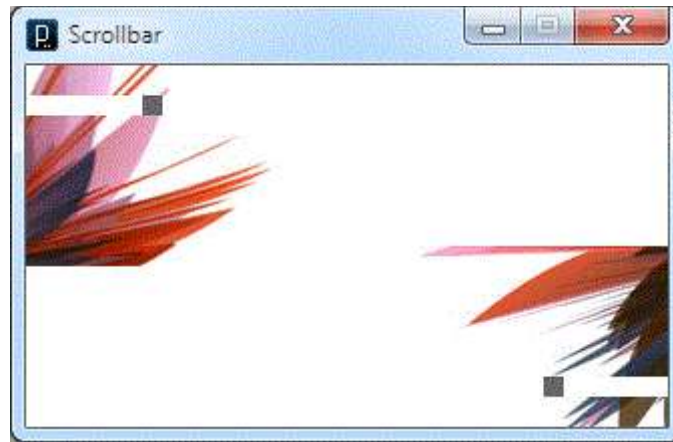


```

int sposMin, sposMax;
int loose;
boolean over;
boolean locked;
float ratio;
HScrollbar (int xp, int yp, int sw, int sh, int l) {
  swidth = sw;
  sheight = sh;
  int widthtoheight = sw - sh;
  ratio = (float)sw / (float)widthtoheight;
  xpos = xp;
  ypos = yp-sheight/2;
  spos = xpos + swidth/2 - sheight/2;
  newspos = spos;
  sposMin = xpos;
  sposMax = xpos + swidth - sheight;
  loose = l;
}
void update() {
  if(over()) {
    over = true;
  } else {
    over = false;
  }
  if(mousePressed && over) {
    locked = true;
  }
  if(!mousePressed) {
    locked = false;
  }
  if(locked) {
    newspos = constrain(mouseX-sheight/2, sposMin, sposMax);
  }
  if(abs(newspos - spos) > 1) {
    spos = spos + (newspos-spos)/loose;
  }
}
int constrain(int val, int minv, int maxv) {
  return min(max(val, minv), maxv);
}
boolean over() {
  if(mouseX > xpos && mouseX < xpos+swidth &&
  mouseY > ypos && mouseY < ypos+sheight) {
    return true;
  } else {
    return false;
  }
}
void display() {
  fill(255);
  rect(xpos, ypos, swidth, sheight);
  if(over || locked) {
    fill(153, 102, 0);
  } else {

```

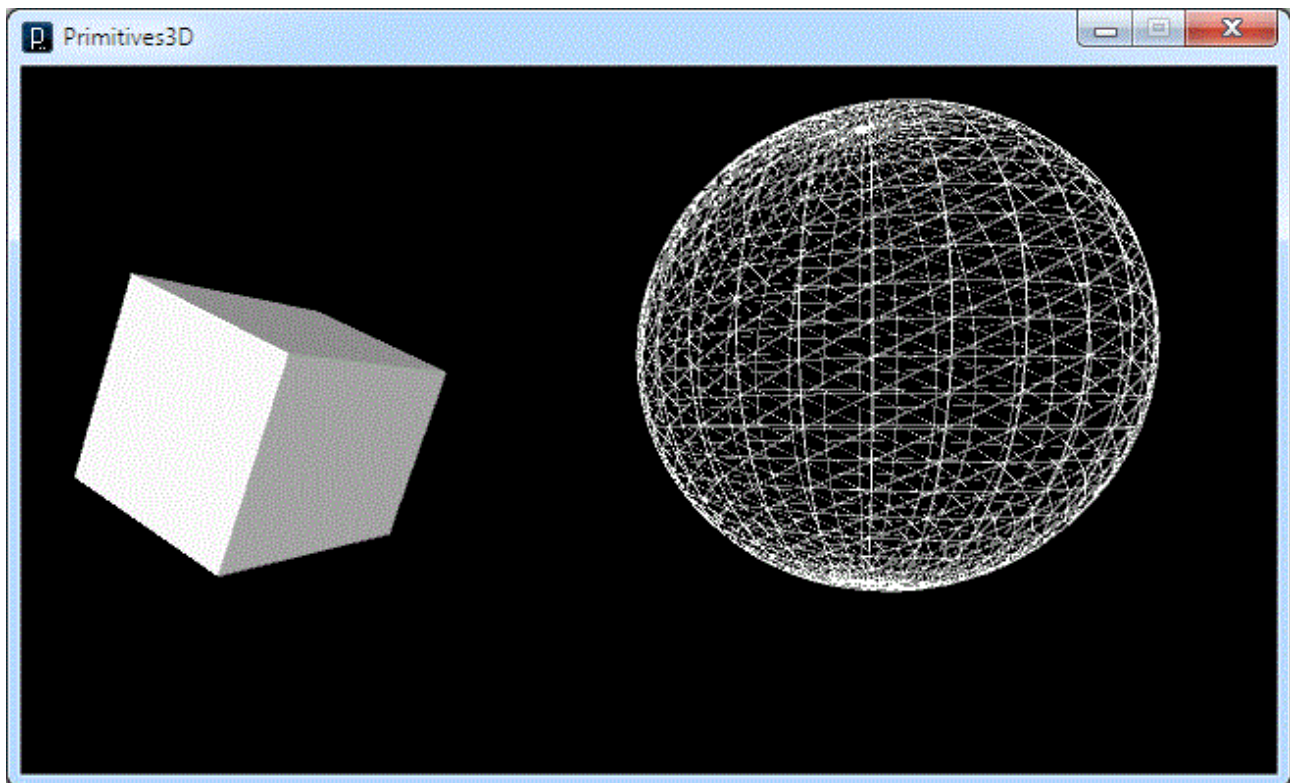
```
    fill(102, 102, 102);  
  }  
  rect(spos, ypos, sheight, sheight);  
}  
float getPos() {  
  return spos * ratio;  
}  
}
```



3D

Esempio, primitive 3D: cubo e sfera.

```
size(640, 360, P3D);  
background(0);  
lights();  
noStroke();  
pushMatrix();  
translate(130, height/2, 0);  
rotateY(1.25);  
rotateX(-0.4);  
box(100);  
popMatrix();  
noFill();  
stroke(255);  
pushMatrix();  
translate(500, height*0.35, -200);  
sphere(190);  
popMatrix();
```



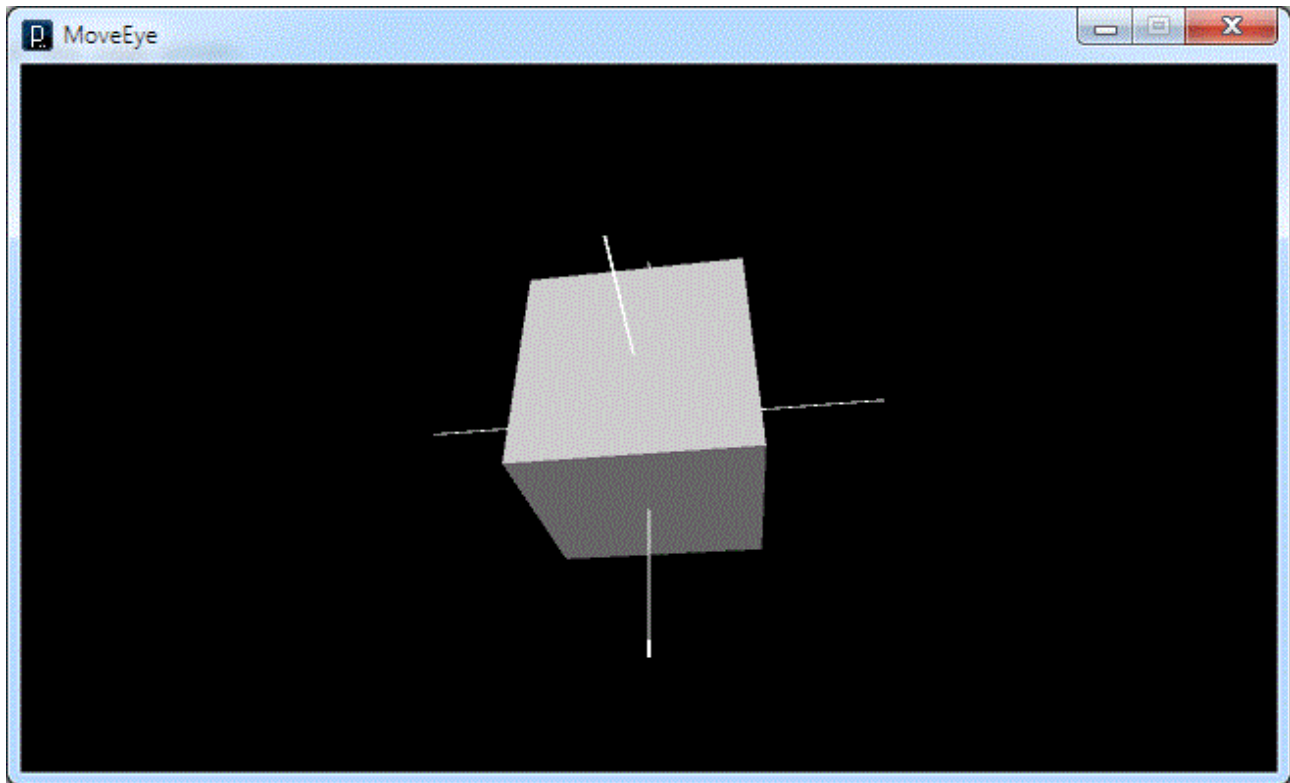
Esempio, videocamera controllata dal mouse.

```
void setup() {  
  size(640, 360, P3D);  
  fill(204);  
}  
void draw() {  
  lights();
```

```

background(0);
camera(30.0, mouseY, 220.0,           // eyeX, eyeY, eyeZ
      0.0, 0.0, 0.0,                 // centerX, centerY, centerZ
      0.0, 1.0, 0.0);                // upX, upY, upZ
noStroke();
box(90);
stroke(255);
line(-100, 0, 0, 100, 0, 0);
line(0, -100, 0, 0, 100, 0);
line(0, 0, -100, 0, 0, 100);
}

```



Esempio, di prospettiva.

```

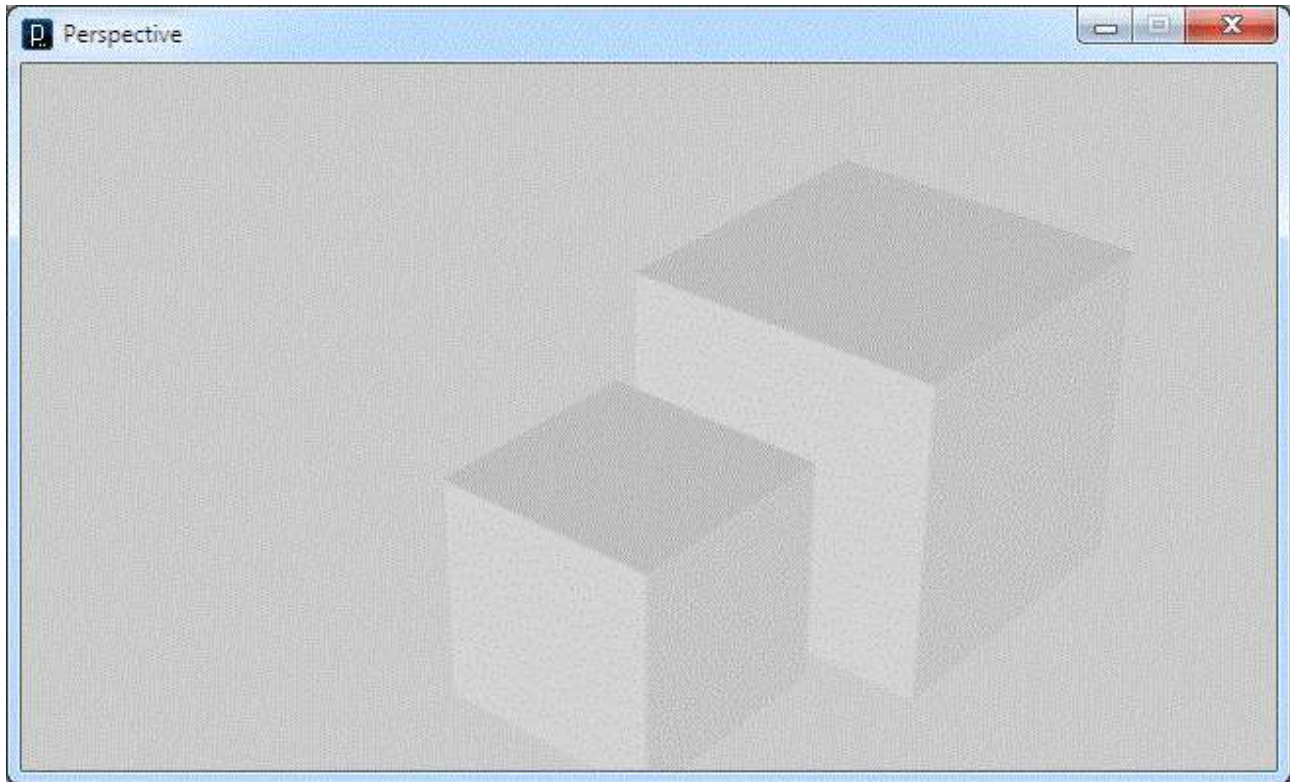
void setup() {
  size(640, 360, P3D);
  noStroke();
}
void draw() {
  lights();
  background(204);
  float cameraY = height/2.0;
  float fov = mouseX/float(width) * PI/2;
  float cameraZ = cameraY / tan(fov / 2.0);
  float aspect = float(width)/float(height);
  if (mousePressed) {
    aspect = aspect / 2.0;
  }
  perspective(fov, aspect, cameraZ/10.0, cameraZ*10.0);
  translate(width/2+30, height/2, 0);
  rotateX(-PI/6);
}

```

```

rotateY(PI/3 + mouseY/float(height) * PI);
box(45);
translate(0, 0, -50);
box(30);
}

```



Esempio, simulare uno stormo di uccelli.

```

int birdCount = 200;
Bird[] birds = new Bird[birdCount];
float[] x = new float[birdCount];
float[] y = new float[birdCount];
float[] z = new float[birdCount];
float[] rx = new float[birdCount];
float[] ry = new float[birdCount];
float[] rz = new float[birdCount];
float[] spd = new float[birdCount];
float[] rot = new float[birdCount];
void setup() {
  size(640, 360, P3D);
  noStroke();
  // inizializzazione array con valori random
  for (int i = 0; i < birdCount; i++){
    birds[i] = new Bird(random(-300, 300), random(-300, 300),
      random(-500, -2500), random(5, 30), random(5, 30));
    x[i] = random(20, 340);
    y[i] = random(30, 350);
    z[i] = random(1000, 4800);
    rx[i] = random(-160, 160);
    ry[i] = random(-55, 55);
    rz[i] = random(-20, 20);
  }
}

```

```

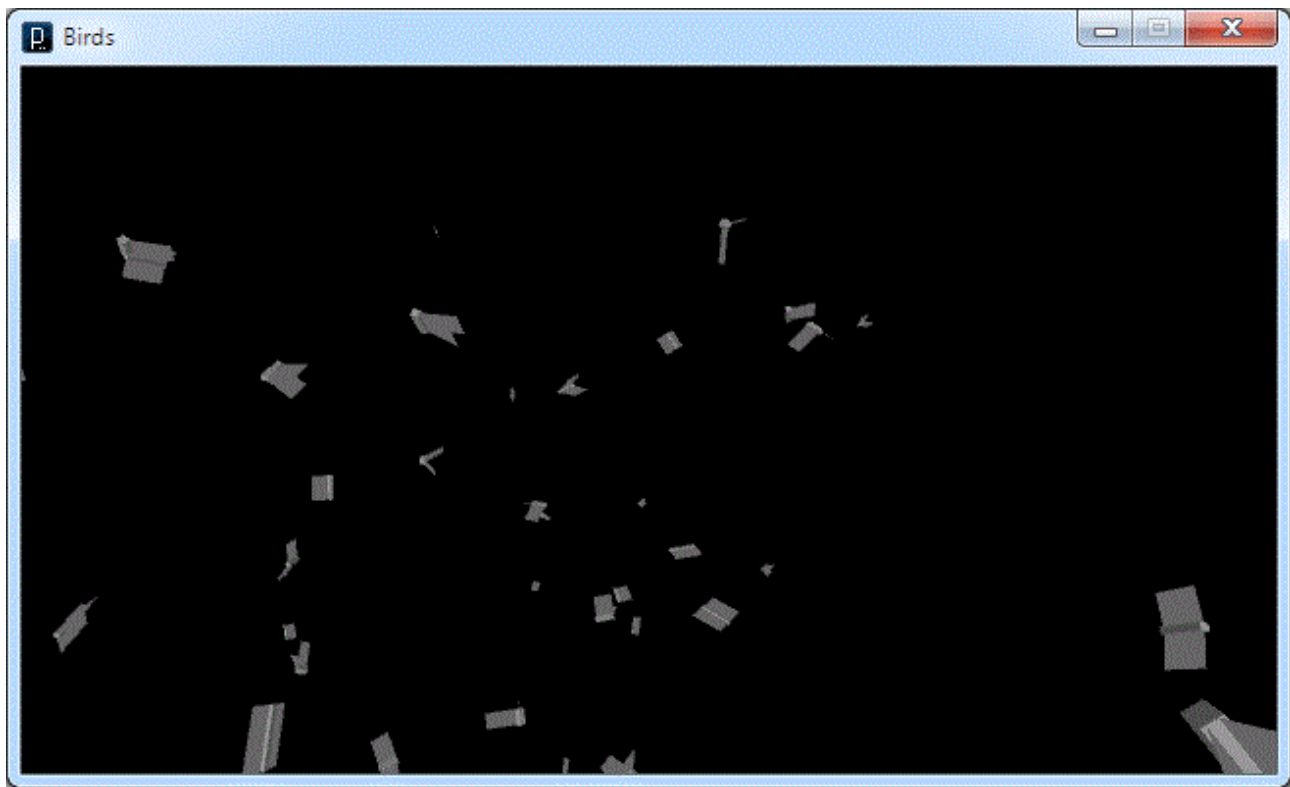
    spd[i] = random(.1, 3.75);
    rot[i] = random(.025, .15);
}
}
void draw() {
    background(0);
    lights();
    for (int i = 0; i < birdCount; i++){
        birds[i].setFlight(x[i], y[i], z[i], rx[i], ry[i], rz[i]);
        birds[i].setWingSpeed(spd[i]);
        birds[i].setRotSpeed(rot[i]);
        birds[i].fly();
    }
}
class Bird {
    // proprietà
    float offsetX, offsetY, offsetZ;
    float w, h;
    int bodyFill;
    int wingFill;
    float ang = 0, ang2 = 0, ang3 = 0, ang4 = 0;
    float radiusX = 120, radiusY = 200, radiusZ = 700;
    float rotX = 15, rotY = 10, rotZ = 5;
    float flapSpeed = 0.4;
    float rotSpeed = 0.1;
    // costruttore
    Bird(){
        this(0, 0, 0, 60, 80);
    }
    Bird(float offsetX, float offsetY, float offsetZ,
        float w, float h){
        this.offsetX = offsetX;
        this.offsetY = offsetY;
        this.offsetZ = offsetZ;
        this.h = h;
        this.w = w;
        bodyFill = color(153);
        wingFill = color(204);
    }
    void setFlight(float radiusX, float radiusY, float radiusZ,
        float rotX, float rotY, float rotZ){
        this.radiusX = radiusX;
        this.radiusY = radiusY;
        this.radiusZ = radiusZ;
        this.rotX = rotX;
        this.rotY = rotY;
        this.rotZ = rotZ;
    }
    void setWingSpeed(float flapSpeed){
        this.flapSpeed = flapSpeed;
    }
    void setRotSpeed(float rotSpeed){
        this.rotSpeed = rotSpeed;
    }
}

```

```

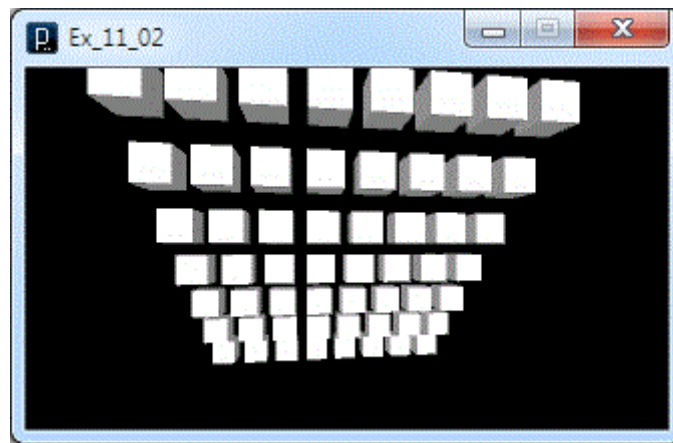
void fly() {
  pushMatrix();
  float px, py, pz;
  // volo
  px = sin(radians(ang3)) * radiusX;
  py = cos(radians(ang3)) * radiusY;
  pz = sin(radians(ang4)) * radiusZ;
  translate(width/2 + offsetX + px, height/2 + offsetY+py, -700 + offsetZ+pz);
  rotateX(sin(radians(ang2)) * rotX);
  rotateY(sin(radians(ang2)) * rotY);
  rotateZ(sin(radians(ang2)) * rotZ);
  fill(bodyFill);
  box(w/5, h, w/5);
  // ala sinistra
  fill(wingFill);
  pushMatrix();
  rotateY(sin(radians(ang)) * 20);
  rect(0, -h/2, w, h);
  popMatrix();
  // ala destra
  pushMatrix();
  rotateY(sin(radians(ang)) * -20);
  rect(-w, -h/2, w, h);
  popMatrix();
  // flap
  ang += flapSpeed;
  if (ang > 3) {
    flapSpeed*=-1;
  }
  if (ang < -3) {
    flapSpeed*=-1;
  }
  ang2 += rotSpeed;
  ang3 += 1.25;
  ang4 += 0.55;
  popMatrix();
}
}

```

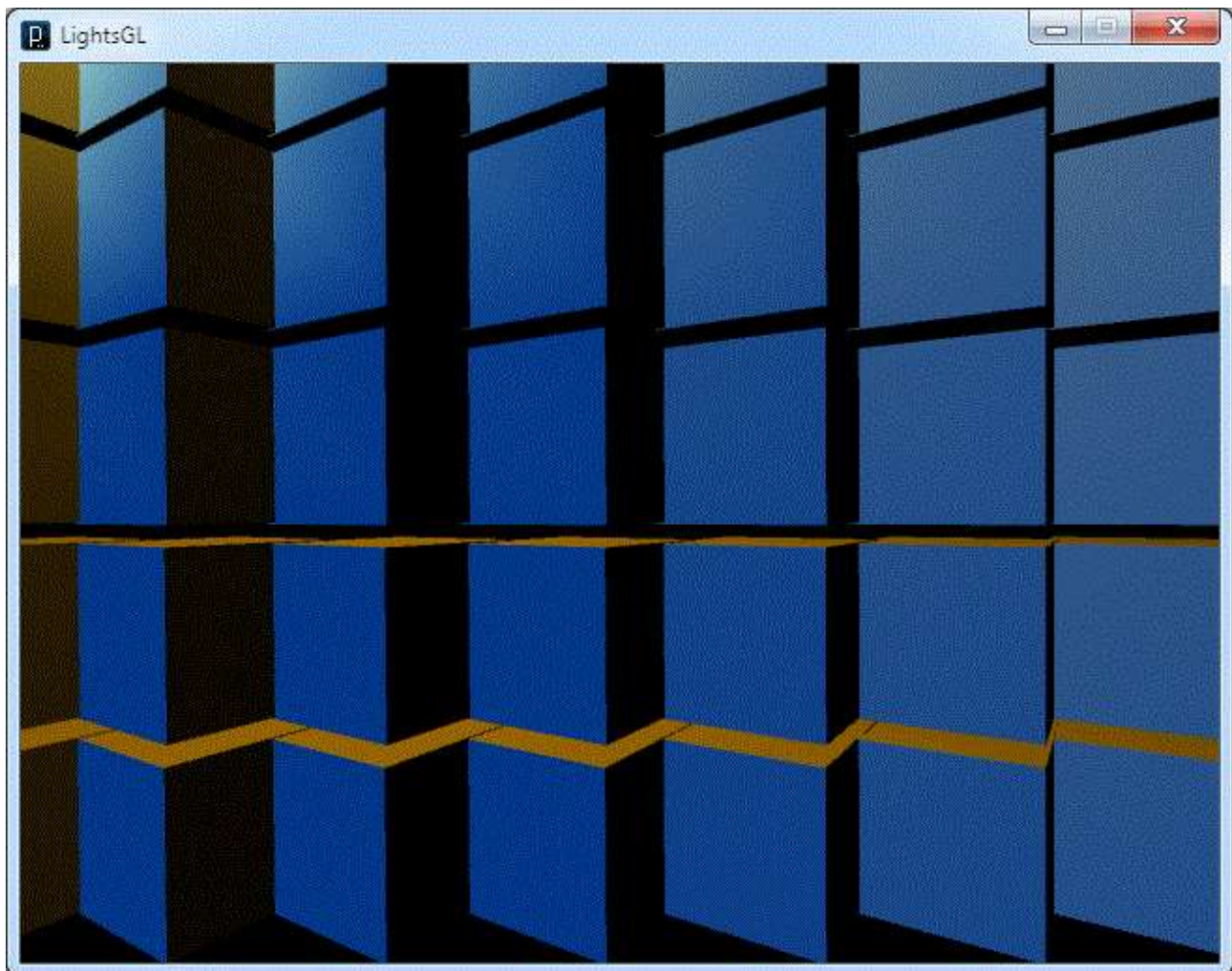
Esempio, **OpenGL** (*Open Graphics Library*).

```
import processing.opengl.*;
void setup() {
  size(320, 180, OPENGL);
  noStroke();
  fill(255);
}
void draw() {
  lights();
  rotateY(PI/24);
  background(0);
  translate(width/2, height/2, -20);
  int dim = 18;
  for (int i = -height/2; i < height/2; i += dim*1.4) {
    for (int j = -height/2; j < height/2; j += dim*1.4) {
      pushMatrix();
      translate(i, j, -j);
      box(dim, dim, dim);
      popMatrix();
    }
  }
}
```



Esempio.

```
import processing.opengl.*;
void setup()
{
  size(640, 480, OPENGL);
  noStroke();
}
void draw()
{
  defineLights();
  background(0);
  for (int x = 0; x <= width; x += 100) {
    for (int y = 0; y <= height; y += 100) {
      pushMatrix();
      translate(x, y);
      rotateY(map(mouseX, 0, width, 0, PI));
      rotateX(map(mouseY, 0, height, 0, PI));
      box(90);
      popMatrix();
    }
  }
}
void defineLights() {
  // punto luce orange sulla destra
  pointLight(150, 100, 0,           // colore
            200, -150, 0);          // posizione
  // luce direzionale blue dalla sinistra
  directionalLight(0, 102, 255,      // colore
                  1, 0, 0);          // assi x-, y-, z
  // luce yellow davanti
  spotLight(255, 255, 109,           // colore
            0, 40, 200,              // posizione
            0, -0.5, -0.5,           // direzione
            PI / 2, 2);              // angolo
}
```



Esempio, progettare una sfera che ruota.

```
import processing.opengl.*;
PImage bg;
PImage texmap;
int sDetail = 35; // Sphere detail setting
float rotationX = 0;
float rotationY = 0;
float velocityX = 0;
float velocityY = 0;
float globeRadius = 225;
float pushBack = 0;
float[] cx, cz, sphereX, sphereY, sphereZ;
float sinLUT[];
float cosLUT[];
float SINCOS_PRECISION = 0.5;
int SINCOS_LENGTH = int(360.0 / SINCOS_PRECISION);
void setup() {
  size(640, 480, OPENGLE);
  texmap = loadImage("world32k.jpg");
  initializeSphere(sDetail);
}
void draw() {
  background(0);
  renderGlobe();
}
```

```

}
void renderGlobe() {
  pushMatrix();
  translate(width/2.0, height/2.0, pushBack);
  pushMatrix();
  noFill();
  stroke(255,200);
  strokeWeight(2);
  smooth();
  popMatrix();
  lights();
  pushMatrix();
  rotateX( radians(-rotationX) );
  rotateY( radians(270 - rotationY) );
  fill(200);
  noStroke();
  textureMode(IMAGE);
  texturedSphere(globeRadius, texmap);
  popMatrix();
  popMatrix();
  rotationX += velocityX;
  rotationY += velocityY;
  velocityX *= 0.95;
  velocityY *= 0.95;
  if(mousePressed){
    velocityX += (mouseY-pmouseY) * 0.01;
    velocityY -= (mouseX-pmouseX) * 0.01;
  }
}
void initializeSphere(int res)
{
  sinLUT = new float[SINCOS_LENGTH];
  cosLUT = new float[SINCOS_LENGTH];

  for (int i = 0; i < SINCOS_LENGTH; i++) {
    sinLUT[i] = (float) Math.sin(i * DEG_TO_RAD * SINCOS_PRECISION);
    cosLUT[i] = (float) Math.cos(i * DEG_TO_RAD * SINCOS_PRECISION);
  }

  float delta = (float)SINCOS_LENGTH/res;
  float[] cx = new float[res];
  float[] cz = new float[res];
  for (int i = 0; i < res; i++) {
    cx[i] = -cosLUT[(int) (i*delta) % SINCOS_LENGTH];
    cz[i] = sinLUT[(int) (i*delta) % SINCOS_LENGTH];
  }

  int vertCount = res * (res-1) + 2;
  int currVert = 0;
  sphereX = new float[vertCount];
  sphereY = new float[vertCount];
  sphereZ = new float[vertCount];
  float angle_step = (SINCOS_LENGTH*0.5f)/res;
  float angle = angle_step;

```

```

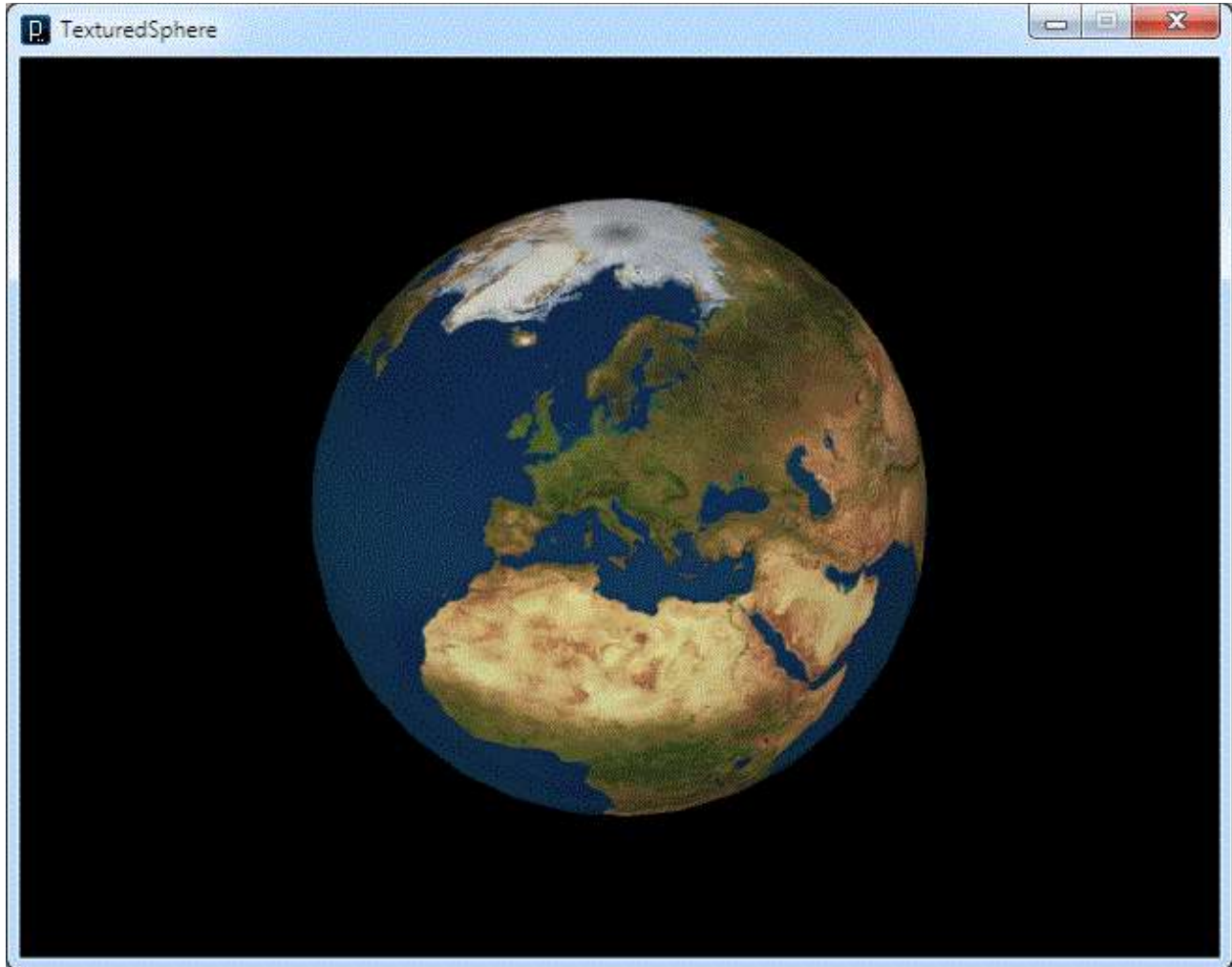
for (int i = 1; i < res; i++) {
    float curradius = sinLUT[(int) angle % SINCOS_LENGTH];
    float currY = -cosLUT[(int) angle % SINCOS_LENGTH];
    for (int j = 0; j < res; j++) {
        sphereX[currVert] = cx[j] * curradius;
        sphereY[currVert] = currY;
        sphereZ[currVert++] = cz[j] * curradius;
    }
    angle += angle_step;
}
sDetail = res;
}
void texturedSphere(float r, PImage t)
{
    int v1,v11,v2;
    r = (r + 240 ) * 0.33;
    beginShape(TRIANGLE_STRIP);
    texture(t);
    float iu=(float)(t.width-1)/(sDetail);
    float iv=(float)(t.height-1)/(sDetail);
    float u=0,v=iv;
    for (int i = 0; i < sDetail; i++) {
        vertex(0, -r, 0,u,0);
        vertex(sphereX[i]*r, sphereY[i]*r, sphereZ[i]*r, u, v);
        u+=iu;
    }
    vertex(0, -r, 0,u,0);
    vertex(sphereX[0]*r, sphereY[0]*r, sphereZ[0]*r, u, v);
    endShape();
    int voff = 0;
    for(int i = 2; i < sDetail; i++) {
        v1=v11=voff;
        voff += sDetail;
        v2=voff;
        u=0;
        beginShape(TRIANGLE_STRIP);
        texture(t);
        for (int j = 0; j < sDetail; j++) {
            vertex(sphereX[v1]*r, sphereY[v1]*r, sphereZ[v1]*r, u, v);
            vertex(sphereX[v2]*r, sphereY[v2]*r, sphereZ[v2]*r, u, v+iv);
            u+=iu;
        }
        v1=v11;
        v2=voff;
        vertex(sphereX[v1]*r, sphereY[v1]*r, sphereZ[v1]*r, u, v);
        vertex(sphereX[v2]*r, sphereY[v2]*r, sphereZ[v2]*r, u, v+iv);
        endShape();
        v+=iv;
    }
    u=0;
    beginShape(TRIANGLE_STRIP);
    texture(t);
    for (int i = 0; i < sDetail; i++) {
        v2 = voff + i;

```

```

    vertex(sphereX[v2]*r, sphereY[v2]*r, sphereZ[v2]*r, u, v);
    vertex(0, r, 0,u,v+iv);
    u+=iu;
}
vertex(sphereX[voff]*r, sphereY[voff]*r, sphereZ[voff]*r, u, v);
endShape();
}

```



I/O

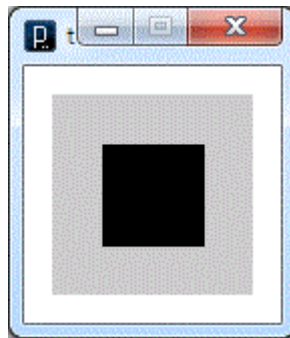
TASTIERA

keyPressed

La variabile di sistema *keyPressed* vale *true* se è premuto un tasto qualsiasi, *false* se non è premuto alcun tasto.

Esempio.

```
/* tastiera_01
   premendo un qualsiasi tasto, il quadrato che normalmente e'
   visualizzato con riempimento colore bianco, diventa colore nero */
void draw()
{
  if ( keyPressed == true )
    fill(0);                // riempimento colore nero
  else
    fill(255);              // riempimento colore bianco
  rect(25, 25, 50, 50);
}
```



key

La variabile di sistema *key* contiene l'ultimo tasto premuto.

Per i tasti con valori non compresi nel codice **ASCII** (*American Standard Code for Information Interchange*), occorre usare la variabile di sistema *keyCode*.

Esempio.

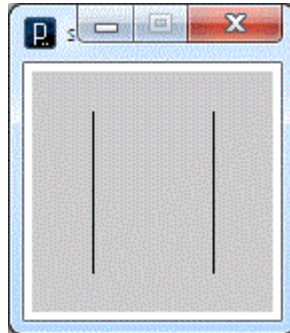
```
/* tastiera_02
   disegna la lettera h oppure n alla pressione del tasto relativo
   alla lettera */
void setup() {
  size(120, 120);
  smooth();
}
void draw() {
  background(204);
  if (keyPressed)
  {
    if ((key == 'h') || (key == 'H'))
```



```

    line(30, 60, 90, 60);
    if ((key == 'n') || (key == 'N'))
        line(30, 20, 90, 100);
    }
    line(30, 20, 30, 100);
    line(90, 20, 90, 100);
}

```



keyCode

La variabile di sistema `keyCode` è usata per il controllo dei tasti speciali quali UP, DOWN, LEFT, RIGHT, ALT, CONTROL, SHIFT.

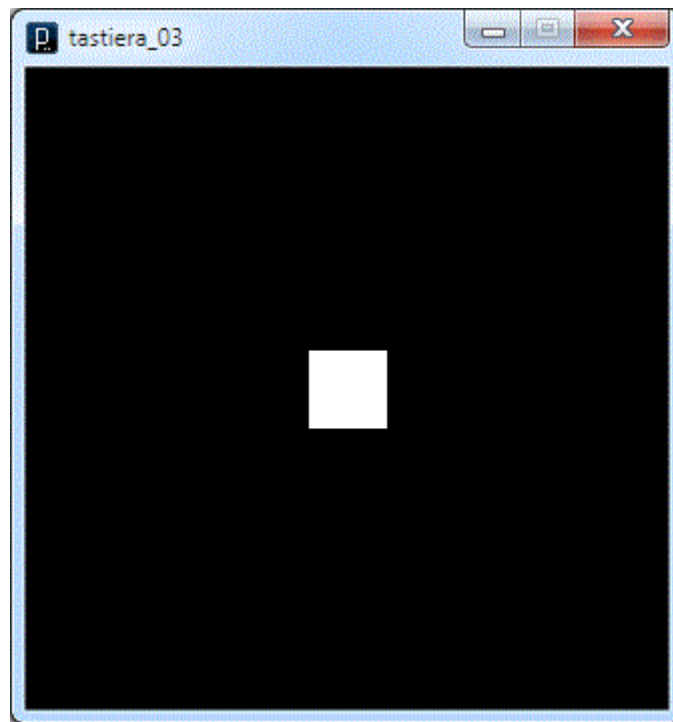
Per il controllo di questi tasti speciali è necessario controllare preventivamente se il tasto premuto è codificato tra quelli speciali gestiti da `keyCode`; per fare questo si usa la condizione `if (key == CODED)` come mostrato nell'esempio seguente.

Esempio.

```

/* tastiera_03
   muove il quadrato a sinistra, destra, alto, basso premendo
   rispettivamente i tasti LEFT, RIGHT, UP, DOWN */
int x = 140, y=140;
void setup() {
    size(320, 320);
}
void draw() {
    background(0);
    if (keyPressed && (key == CODED))
    {
        if ( keyCode == LEFT )
            x--;
        if ( keyCode == RIGHT )
            x++;
        if ( keyCode == UP )
            y--;
        if ( keyCode == DOWN )
            y++;
    }
    rect(x, y, 40, 40);
}

```



MOUSE

Processing mette a disposizione variabili di sistema e funzioni connesse al funzionamento del mouse e al verificarsi di un evento a esso correlato.

mouseX, mouseY

Contengono le coordinate del mouse nel frame corrente.

pmouseX, pmouseY

Contengono le coordinate del mouse nel frame precedente.

mouseButton

Il valore della variabile *mouseButton* può essere *LEFT*, *RIGHT* o *CENTER* secondo il pulsante premuto.

Inoltre, Processing mette a disposizione alcuni eventi connessi al funzionamento del mouse.

mousePressed

Restituisce true se il mouse è correntemente premuto.

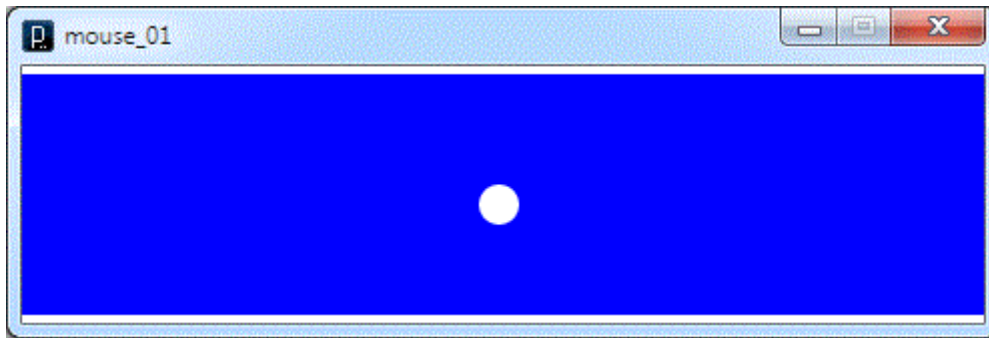
Esempio.

```
/* mouse_01
   il cerchio, con riempimento di colore bianco, segue il movimento
   del mouse; alla pressione di un qualsiasi tasto del mouse il
   riempimento diventa colore nero, per ritornare bianco al rilascio
   del tasto */
void setup() {
  size(480, 120);
  smooth();                // arrotonda i bordi
  noStroke();              // nessun bordo
}
void draw() {
  background(0,0,255);    // sfondo colore blu
```

```

ellipse(mouseX, mouseY, 20,20);
if (mousePressed)
    fill(0);                // riempimento colore nero
    else
    fill(255);              // riempimento colore bianco
}

```

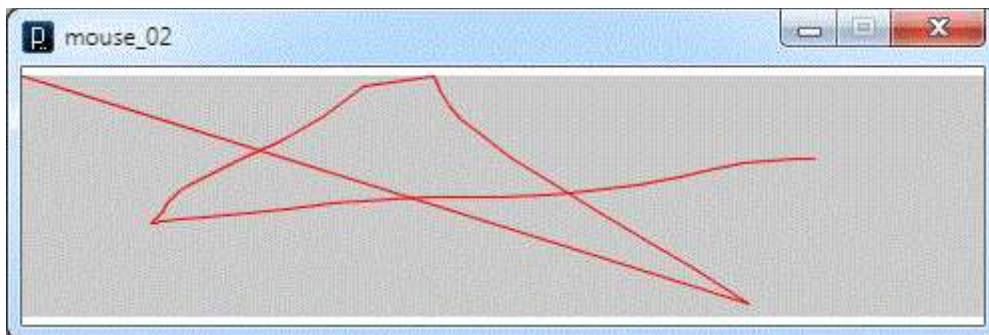


Esempio.

```

/* mouse_02
   disegna una linea seguendo il movimento del mouse */
void setup() {
    size(480, 120);
    stroke(255, 0, 0);      // colore linea rosso
}
void draw() {
    line(mouseX, mouseY, pmouseX, pmouseY);
}

```



Esempio.

```

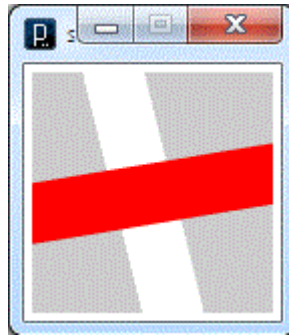
/* mouse_03
   alla pressione del tasto sx mouse disegna una linea colore rosso,
   alla pressione del tasto dx mouse disegna una linea colore verde,
   alla pressione del tasto centrale mouse disegna una linea colore blu */
void setup() {
    size(120, 120);
    smooth();                // arrotonda i bordi
    strokeWeight(30);        // spessore linea 30 pixel
}
void draw() {
    background(204);
    stroke(255);              // riempimento colore bianco
    line(40, 0, 70, height);
}

```

```

if (mousePressed)
{
  if (mouseButton == LEFT)
    stroke(255,0,0);           // bordo colore rosso
  if (mouseButton == RIGHT)
    stroke(0,255,0);           // bordo colore verde
  if (mouseButton == CENTER)
    stroke(0,0,255);           // bordo colore blu
  line(0, 70, width, 50);
}
}

```



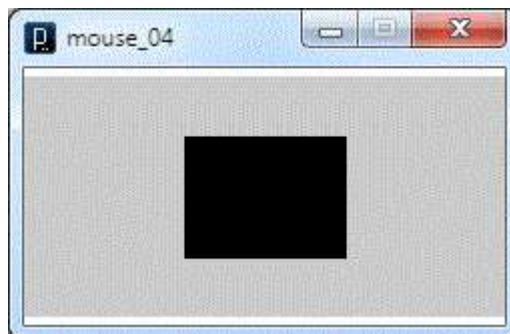
Esempio.

```

/* mouse_04
   disegna un rettangolo con riempimento bianco; passando con il mouse
   all'interno del rettangolo, questo viene colorato di nero */
int x = 80, y = 30, w = 80, h = 60;

void setup() {
  size(240, 120);
}
void draw() {
  background(204);
  if ( (mouseX > x) && (mouseX < x+w) &&
        (mouseY > y) && (mouseY < y+h) )
    fill(0);           // riempimento colore nero
  else
    fill(255);         // riempimento colore bianco
  rect(x, y, w, h);
}

```



RS232

Esempio, codice per la scheda Arduino.

```
int sensorPin = 0;           // seleziona l'input pin
int val = 0;
void setup() {
  Serial.begin(9600);        // apre la porta seriale
}
void loop() {
  val = analogRead(sensorPin) / 4; // legge il valore dal sensore
  Serial.print(val, BYTE);
  delay(100);                // attesa 100 millisecondi
}
```

Esempio, codice per la scheda Arduino.

```
import processing.serial.*;
Serial port;                  // crea un oggetto dalla classe Serial
float val;                    // dato ricevuto dalla porta seriale
int x;
float easing = 0.05;
float easedVal;
void setup() {
  size(440, 440);
  frameRate(30);
  smooth();
  String arduinoPort = Serial.list()[0];
  port = new Serial(this, arduinoPort, 9600);
  background(0);
}
void draw() {
  if (port.available() > 0) { // se il dato è disponibile
    val = port.read();        // lo legge e lo memorizza in val
    val = map(val, 0, 255, 0, height); // converte il valore
  }
  rect(40, val-10, 360, 20);
}
float targetVal = val;
easedVal += (targetVal - easedVal) * easing;
stroke(0);
line(x, 0, x, height);
stroke(255);
line(x+1, 0, x+1, height);
line(x, 220, x, val);
line(x, 440, x, easedVal + 220); // valore medio
x++;
if (x > width) {
  x = 0;
}
}
```

Esempio, inviare un carattere ASCII (byte valore 65) per la scheda Arduino.

```
int firstSensor = 0;           // primo sensore analogico
int secondSensor = 0;         // secondo sensore analogico
int thirdSensor = 0;          // sensore digitale
int inByte = 0;
void setup()
{
  // start porta seriale a 9600 bps:
  Serial.begin(9600);
  pinMode(2, INPUT);          // il sensore digitale è sul pin 2
  establishContact();
}
void loop()
{
  if (Serial.available() > 0) {
    inByte = Serial.read();
    // legge il primo input analogico e divide per 4, range 0-255
    firstSensor = analogRead(0)/4;
    // delay 10 ms per memorizzare il dato nell'ADC
    delay(10);
    // legge il secondo input analogico, divide per 4, range 0-255:
    secondSensor = analogRead(1)/4;
    thirdSensor = 100 + (155 * digitalRead(2));
    Serial.print(firstSensor, BYTE);
    Serial.print(secondSensor, BYTE);
    Serial.print(thirdSensor, BYTE);
  }
}
void establishContact() {
  while (Serial.available() <= 0) {
    Serial.print('A', BYTE);
    delay(300);
  }
}
```

Esempio, inviare un byte sulla porta seriale e leggere tre byte in input.

```
import processing.serial.*;
int bgcolor;
int fgcolor;
Serial myPort;
int[] serialInArray = new int[3];
int serialCount = 0;
int xpos, ypos;
boolean firstContact = false;
void setup() {
  size(256, 256);
  noStroke();
  xpos = width/2;
  ypos = height/2;
  // stampa le porte seriali a bordo del sistema
  println(Serial.list());
  String portName = Serial.list()[0];
```

```

    myPort = new Serial(this, portName, 9600);
}
void draw() {
    background(bgcolor);
    fill(fgcolor);
    ellipse(xpos, ypos, 20, 20);
}
void serialEvent(Serial myPort) {
    // legge un byte dalla porta seriale
    int inByte = myPort.read();
    if (firstContact == false) {
        if (inByte == 'A') {
            myPort.clear();                // pulisce il buffer della porta seriale
            firstContact = true;
            myPort.write('A');
        }
    }
    else {
        serialInArray[serialCount] = inByte;
        serialCount++;
        // se ci sono tre byte
        if (serialCount > 2 ) {
            xpos = serialInArray[0];
            ypos = serialInArray[1];
            fgcolor = serialInArray[2];
            // stampa i valori
            println(xpos + "\t" + ypos + "\t" + fgcolor);
            myPort.write('A');
            serialCount = 0;
        }
    }
}
}

```

Esempio, gestione della seriale su linea duplex.

```

import processing.serial.*;
Serial myPort;
int whichKey = -1;
int inByte = -1;
void setup() {
    size(400, 300);
    PFont myFont = createFont(PFont.list()[2], 14);
    textFont(myFont);
    println(Serial.list());
    String portName = Serial.list()[0];
    myPort = new Serial(this, portName, 9600);
}
void draw() {
    background(0);
    text("Last Received: " + inByte, 10, 130);
    text("Last Sent: " + whichKey, 10, 100);
}
void serialEvent(Serial myPort) {
    inByte = myPort.read();
}

```

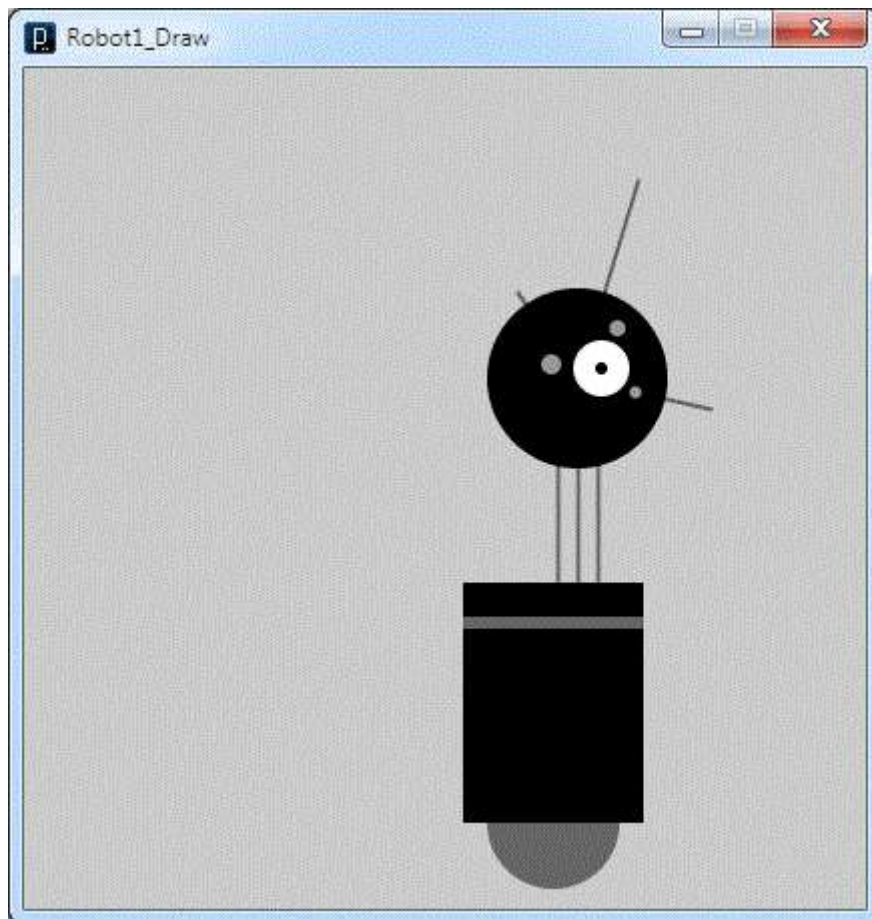


```
}  
void keyPressed() {  
  myPort.write(key);  
  whichKey = key;  
}
```

ROBOT

Esempio, disegnare un robot.

```
size(420, 420);
smooth();
strokeWeight(2);
background(204);
ellipseMode(RADIUS);
// collo
stroke(102)
line(266, 257, 266, 162);           // sinistra
line(276, 257, 276, 162);           // centro
line(286, 257, 286, 162);           // destra
// antenne
line(276, 155, 246, 112);           // piccolo
line(276, 155, 306, 56);            // alto
line(276, 155, 342, 170);           // medio
// corpo
noStroke();
fill(102);
ellipse(264, 377, 33, 33);
fill(0);
rect(219, 257, 90, 120);
fill(102);
rect(219, 274, 90, 6);
// testa
fill(0);
ellipse(276, 155, 45, 45);
fill(255);
ellipse(288, 150, 14, 14);
fill(0);
ellipse(288, 150, 3, 3);
fill(153);
ellipse(263, 148, 5, 5);             // occhio 1
ellipse(296, 130, 4, 4);             // occhio 2
ellipse(305, 162, 3, 3);             // occhio 3
```

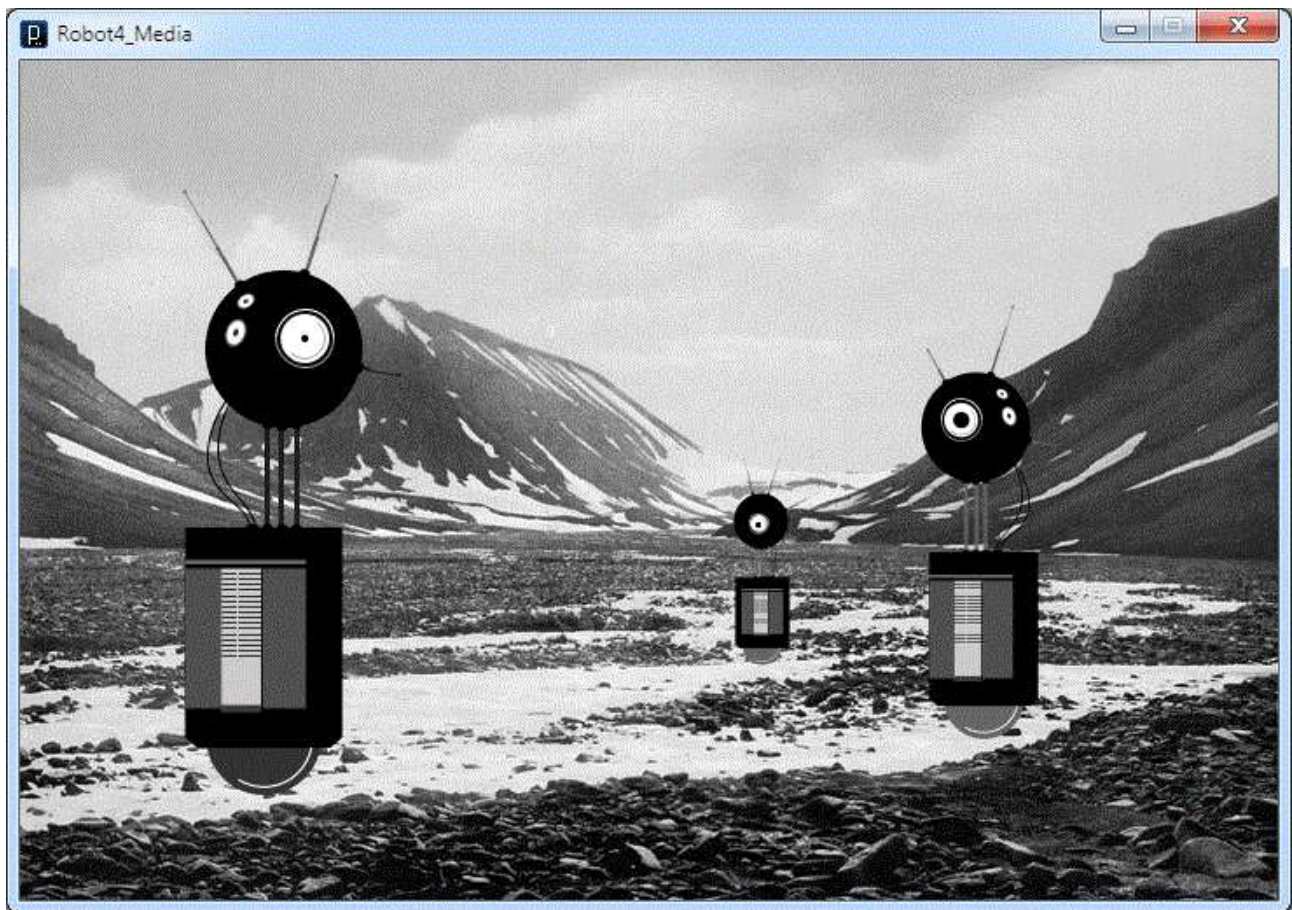


Esempio.

```

PShape bot1;
PShape bot2;
PShape bot3;
PImage landscape;
float easing = 0.05;
float offset = 0;
void setup() {
  size(720, 480);
  bot1 = loadShape("robot1.svg");
  bot2 = loadShape("robot2.svg");
  bot3 = loadShape("robot3.svg");
  landscape = loadImage("alpine.png");
  smooth();
}
void draw() {
  background(landscape);
  float targetOffset = map(mouseY, 0, height, -40, 40);
  offset += (targetOffset - offset) * easing;
  shape(bot1, 85 + offset, 65);
  float smallerOffset = offset * 0.7;
  shape(bot2, 510 + smallerOffset, 140, 78, 248);
  smallerOffset *= -0.5;
  shape(bot3, 410 + smallerOffset, 225, 39, 124);
}

```



MUSICA

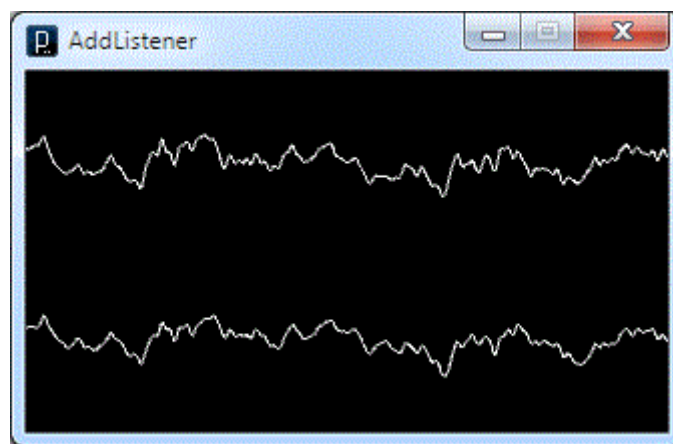
Esempio.

```
import ddf.minim.*;
Minim minim;
AudioPlayer groove;
WaveformRenderer waveform;
void setup()
{
  size(320, 180, P2D);
  minim = new Minim(this);
  groove = minim.loadFile("tango.mp3", 512);
  groove.loop();
  waveform = new WaveformRenderer();
  groove.addListener(waveform);
}
void draw()
{
  background(0);
  waveform.draw();
}
void stop()
{
  groove.close();
  minim.stop();
  super.stop();
}
class WaveformRenderer implements AudioListener
{
  private float[] left;
  private float[] right;
  WaveformRenderer()
  {
    left = null;
    right = null;
  }
  synchronized void samples(float[] samp)
  {
    left = samp;
  }
  synchronized void samples(float[] sampL, float[] sampR)
  {
    left = sampL;
    right = sampR;
  }
  synchronized void draw()
  {
    if ( left != null && right != null )
    {
      noFill();
    }
  }
}
```

```

stroke(255);
beginShape();
for ( int i = 0; i < left.length; i++ )
{
  vertex(i, height/4 + left[i]*50);
}
endShape();
beginShape();
for ( int i = 0; i < right.length; i++ )
{
  vertex(i, 3*(height/4) + right[i]*50);
}
endShape();
}
else if ( left != null )
{
  noFill();
  stroke(255);
  beginShape();
  for ( int i = 0; i < left.length; i++ )
  {
    vertex(i, height/2 + left[i]*50);
  }
  endShape();
}
}
}

```



Esempio, leggere i metadati di un file **MP3** (*Motion Picture Expert Group-1/2 Audio Layer 3*).

```

import ddf.minim.*;
Minim minim;
AudioPlayer groove;
AudioMetaData meta;
void setup()
{
  size(320, 260, P2D);
  minim = new Minim(this);
  groove = minim.loadFile("tango.mp3");
  meta = groove.getMetaData();
}

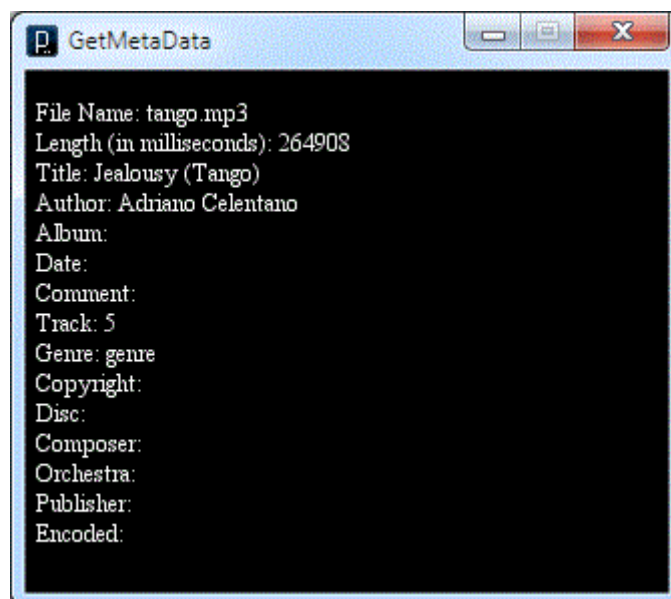
```

```

    textFont(createFont("Serif", 12));
    textMode(SCREEN);
}
int ys = 25;
int yi = 15;
void draw()
{
    background(0);
    int y = ys;
    text("File Name: " + meta.fileName(), 5, y);
    text("Length (in milliseconds): " + meta.length(), 5, y+=yi);
    text("Title: " + meta.title(), 5, y+=yi);
    text("Author: " + meta.author(), 5, y+=yi);
    text("Album: " + meta.album(), 5, y+=yi);
    text("Date: " + meta.date(), 5, y+=yi);
    text("Comment: " + meta.comment(), 5, y+=yi);
    text("Track: " + meta.track(), 5, y+=yi);
    text("Genre: " + meta.genre(), 5, y+=yi);
    text("Copyright: " + meta.copyright(), 5, y+=yi);
    text("Disc: " + meta.disc(), 5, y+=yi);
    text("Composer: " + meta.composer(), 5, y+=yi);
    text("Orchestra: " + meta.orchestra(), 5, y+=yi);
    text("Publisher: " + meta.publisher(), 5, y+=yi);
    text("Encoded: " + meta.encoded(), 5, y+=yi);
}
void stop()
{
    groove.close();
    minim.stop();

    super.stop();
}

```



VIDEO

Esempio.

```
import processing.video.*;
int numPixels;
int[] backgroundPixels;
Capture video;
void setup() {
  size(320, 240, P2D);
  video = new Capture(this, width, height, 24);
  numPixels = video.width * video.height;
  // crea un array per memorizzare l'immagine
  backgroundPixels = new int[numPixels];
  loadPixels();
}
void draw() {
  if (video.available()) {
    video.read(); // legge un nuovo frame video
    video.loadPixels();
    int presenceSum = 0;
    for (int i = 0; i < numPixels; i++) {
      color currColor = video.pixels[i];
      color bkgdColor = backgroundPixels[i];
      // estrae RGB le componenti del pixel corrente
      int currR = (currColor >> 16) & 0xFF;
      int currG = (currColor >> 8) & 0xFF;
      int currB = currColor & 0xFF;
      // estrae RGB le componenti del pixel in background
      int bkgdR = (bkgdColor >> 16) & 0xFF;
      int bkgdG = (bkgdColor >> 8) & 0xFF;
      int bkgdB = bkgdColor & 0xFF;
      int diffR = abs(currR - bkgdR);
      int diffG = abs(currG - bkgdG);
      int diffB = abs(currB - bkgdB);
      presenceSum += diffR + diffG + diffB;
      pixels[i] = 0xFF000000 | (diffR << 16) | (diffG << 8) | diffB;
    }
    updatePixels(); // Notify that the pixels[] array has changed
    println(presenceSum);
  }
}

// quando è premuto un tasto, cattura l'immagine di background in backgroundPixels
void keyPressed() {
  video.loadPixels();
  arraycopy(video.pixels, backgroundPixels);
}
```

Esempio, riprodurre un file **AVI** (*Audio Video Interleave*).

```
import processing.video.*;
Movie myMovie;
void setup() {
  size(320, 240, P2D);
  background(0);
  // carica ed esegue il video
  myMovie = new Movie(this, "sci.avi");
  myMovie.loop();
}
void movieEvent(Movie myMovie) {
  myMovie.read();
}
void draw() {
  tint(255, 20);
  image(myMovie, mouseX-myMovie.width/2, mouseY-myMovie.height/2);
}
```

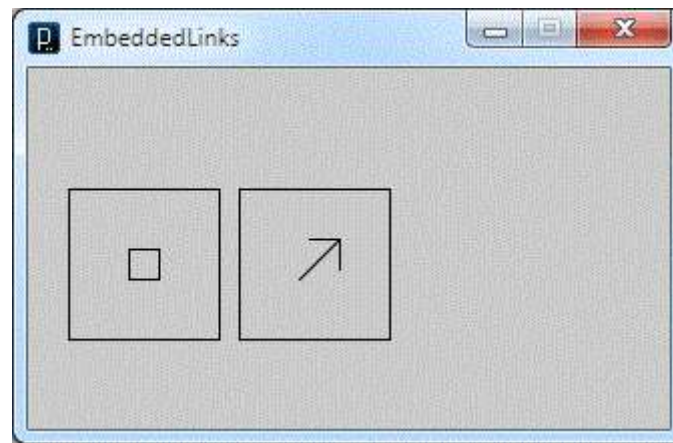


HTML 5.0

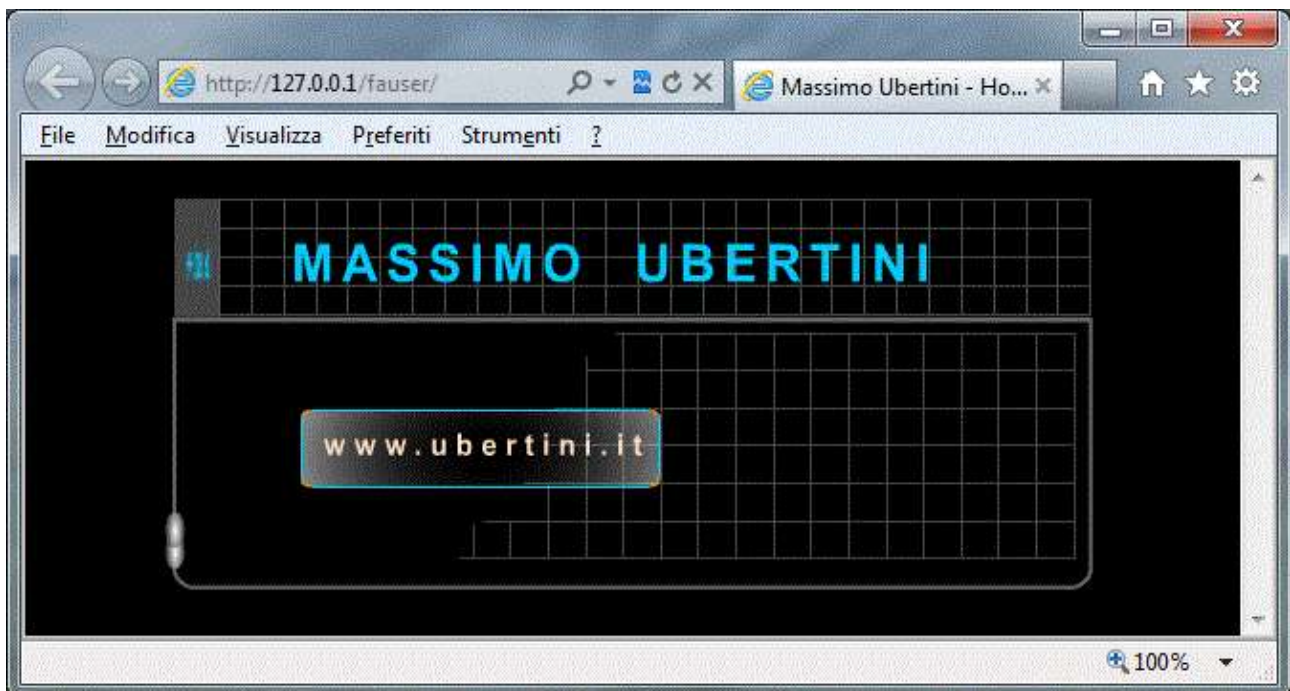
Esempio, aprire un **URL** (*Uniform Resource Locator*).

```
boolean overLeftButton = false;
boolean overRightButton = false;
void setup() {
  size(320, 180);
}
void draw() {
  background(204);
  // pulsante sinistro
  if (overLeftButton == true) {
    fill(255);
  } else {
    noFill();
  }
  rect(20, 60, 75, 75);
  rect(50, 90, 15, 15);
  // pulsante destro
  if (overRightButton == true) {
    fill(255);
  } else {
    noFill();
  }
  rect(105, 60, 75, 75);
  line(135, 105, 155, 85);
  line(140, 85, 155, 85);
  line(155, 85, 155, 100);
}
void mousePressed() {
  if (overLeftButton) {
    link("http://127.0.0.1");
  } else if (overRightButton) {
    link("http://127.0.0.1/fauser/");
  }
}
void mouseMoved() {
  checkButtons();
}
void mouseDragged() {
  checkButtons();
}
void checkButtons() {
  if (mouseX > 20 && mouseX < 95 && mouseY > 60 && mouseY < 135) {
    overLeftButton = true;
  } else if (mouseX > 105 && mouseX < 180 && mouseY > 60 && mouseY < 135) {
    overRightButton = true;
  } else {
    overLeftButton = overRightButton = false;
  }
}
```

}



Fare clic sul pulsante destro, si apre il browser.



Esempio, utilizzare il servizio di geolocation.

File HITOYOUTOO.PDE

```
float pos_x, pos_y;  
boolean locationFound = false;  
boolean locationServiceNotAvailable = false;  
String errorMessage = "", addressString = "";  
PImage worldMap;  
void setup ()  
{  
  size(454, 200);  
  textFont(createFont("Arial", 12));  
  worldMap = loadImage("map.png");  
}  
void draw ()  
{  
  background(worldMap);
```

```

if ( locationServiceNotAvailable ) // failed
{
    textAlign(CENTER);
    fill( 255, 0, 0 );
    text( "Something went wrong:\n" + errorMessage,
        width/4, height/4, width/2, height/2 );
}
else if ( !locationFound ) // waiting for location to come in ..
{
    float w = 5 + (sin(frameCount/60.0) + 1) * (height / 3);
    ellipse( width/2, height/2, w, w );
}
else // we have a location! yay!
{
    noStroke();
    fill( 255, 0,0 );
    ellipse(pos_x, pos_y, 7, 7);
    boolean onRightHalf = pos_x > width/2;
    textAlign( onRightHalf ? RIGHT : LEFT );
    text( "Hi! You are (about) here ..\n"+addressString,
        pos_x + (onRightHalf ? -10 : 10),
        pos_y + 3 );
}
}
/* the following functions are called from plain javascript, see .js tab */
void setGeoLocation ( float latitude, float longitude )
{
    // this is a really simplistic (course) "projection",
    // see: http://en.wikipedia.org/wiki/Map\_projection
    pos_x = (180.0 + longitude) * (width / 360.0);
    pos_y = ( 90.0 - latitude ) * (height / 180.0);

    locationFound = true;
}
void geoLocationError ( String message )
{
    locationServiceNotAvailable = true; // bummer!
    errorMessage = message;
}
void setAddressString ( String address )
{
    addressString = "(" + address + ")";
}

```

File GEOLOC.JS

```

document.write( "<script src=\"http://maps.google.com/maps/api/js?sensor=false\" "+
"type=\"text/javascript\"></script>");
window.onload = function () {
    tryFindSketch();
}
function tryFindSketch () {
    var sketch = Processing.getInstanceById("hiToYouToo");
    if ( sketch == undefined )
        return setTimeout(tryFindSketch, 200);           // try again in 0.2 secs
}

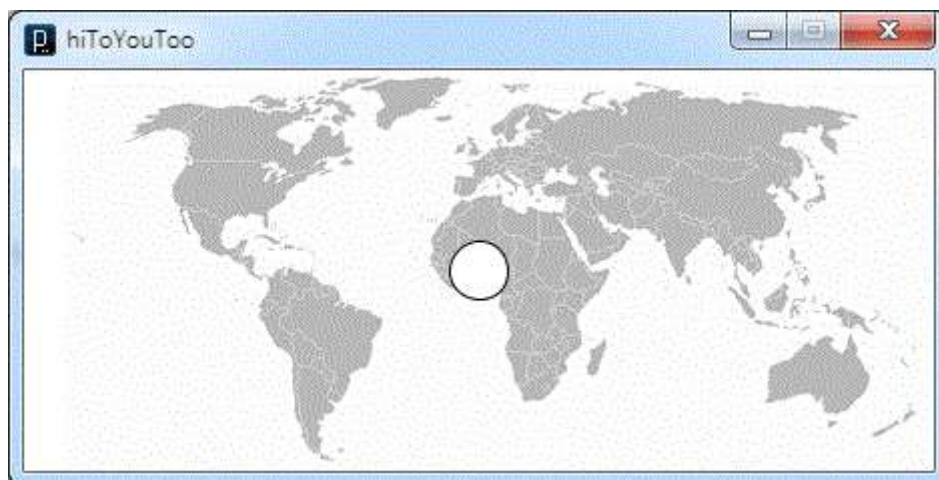
```

```

if ( navigator.geolocation ) {
  navigator.geolocation.getCurrentPosition( function(position) {
    /*success*/
    sketch.setGeoLocation(position.coords.latitude, position.coords.longitude);
    tryReverseGeocoding( sketch, position.coords );
  }, function( position_error ) {
    /*error*/
    sketch.geolocationError(position_error.message);
  });
} else {
  sketch.geolocationError( "Your browser does not support location services." );
}
}

function tryReverseGeocoding ( sketch, latlng ) {
  var geocoder = new google.maps.Geocoder();
  geocoder.geocode({
    'latLng': new google.maps.LatLng(latlng.latitude,latlng.longitude)
  },
  function (data, status) {
    if (status == google.maps.GeocoderStatus.OK) {
      if ( data.length > 0 ) {
        for ( var d in data ) {
          for ( var t in data[d].types ) {
            if ( data[d].types[t] == 'political' ) {
              sketch.setAddressString( data[d].formatted_address );
              return; // done
            }
          }
        }
      }
      sketch.setAddressString( data[0].formatted_address ); // fallback
    }
  } else {
    // ignore ..
  }
});
}

```



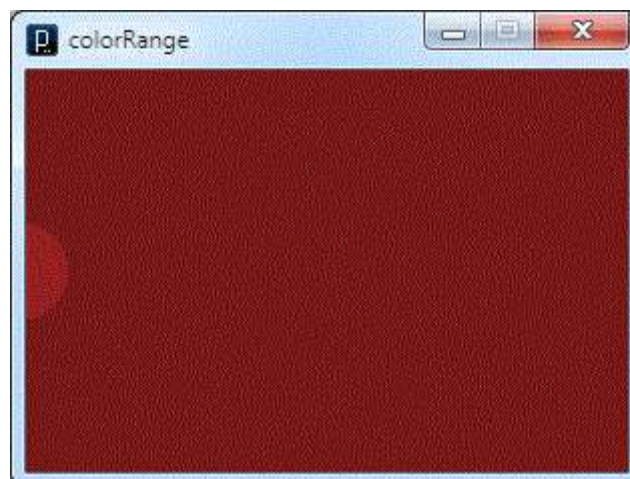
Esempio.

File COLORRANGE.PDE

```
float rangeValue = 0;
float gColorValue = 0;
void setup ()
{
  size(300,200);
  colorMode(HSB);
}
void draw ()
{
  background(gColorValue, 200, 120);
  float c = int(map( rangeValue, 0, 100, 0, 255 ));
  fill( c, 200, 150 );
  stroke( c, 180, 100 );
  float x = map( rangeValue, 1, 100, 0, width );
  ellipse( x, height/2, 50, 50 );
  gColorValue += (c-gColorValue)/30.0;
}
void newRangeValue ( float v )
{
  rangeValue = v;
}
```

File INTERFACE.JS

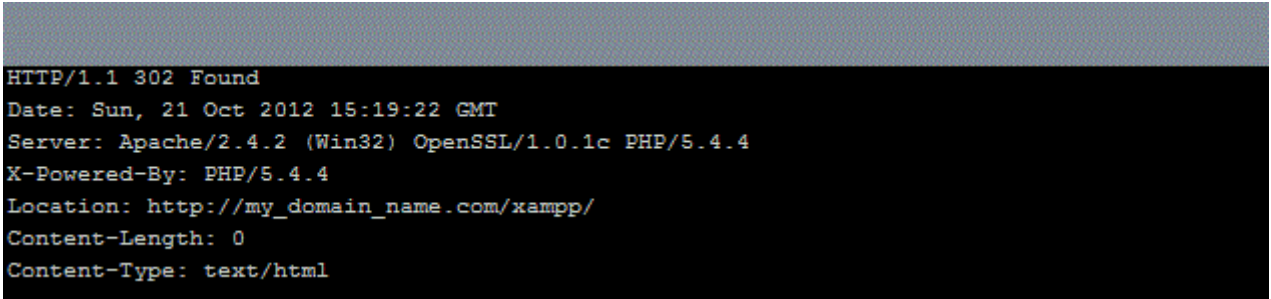
```
window.onload = function () {
  tryFindSketch();
}
function tryFindSketch () {
  var sketch = Processing.getInstanceById(getProcessingSketchID());
  if ( sketch == undefined )
    return setTimeout(tryFindSketch, 200);           // try again ..
  // get slider from DOM
  var range = document.getElementById("form-range");
  // add listener
  range.onchange = function () {
    sketch.newRangeValue( range.value );
  }
}
```



NETWORK

Esempio, progettare un client **HTTP** (*Hyper Text Transfer Protocol*).

```
import processing.net.*;
Client c;
String data;
void setup() {
  size(200, 200);
  background(50);
  fill(200);
  // si connette al server sulla porta 80
  c = new Client(this, "127.0.0.1", 80);
  // usa il comando HTTP GET
  c.write("GET / HTTP/1.1\n");
  c.write("Host: my_domain_name.com\n\n");
}
void draw() {
  if (c.available() > 0) {
    data = c.readString();
    println(data);
  }
}
```



```
HTTP/1.1 302 Found
Date: Sun, 21 Oct 2012 15:19:22 GMT
Server: Apache/2.4.2 (Win32) OpenSSL/1.0.1c PHP/5.4.4
X-Powered-By: PHP/5.4.4
Location: http://my_domain_name.com/xampp/
Content-Length: 0
Content-Type: text/html
```

PIETRANGELO PASQUALE

Dip. Informatica Industriale
I.T.I.S. "Giacomo Fauser"
Via Ricci, 14
28100 Novara Italy
tel. +39 0321482411
fax +39 0321482444
<http://www.fauser.edu>
pietrangelo.pasquale@fauser.edu