A cura di:
Crippa Davide
Califano Marco

Processing è un **linguaggio di programmazione** che consente di sviluppare diverse **applicazioni grafiche** come giochi, animazioni e contenuti interattivi.

È un progetto **OpenSource**[1] sviluppato da due ricercatori del **MIT** (Massachusetts Institute of technology, **istituto di tecnologia del Massachusetts**, una delle più importanti università di ricerca del ***mondo***) da due studenti per la loro tesi di laurea.

Ultimamente inizia a essere usato anche da molti ricercatori per via della sua **immediatezza** e per la **facilità** con la quale è possibile realizzare animazioni, interfacce grafiche e visualizzazioni di dati.

[1]OPENSOURCE: In informatica, si riferisce a un **software** non protetto da copyright e **liberamente modificabile** dagli utenti.

Download e installazione:

Download Processing. Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below.

3.3.6 (4 September 2017)

Windows 64-bit
Windows 32-bit
Linux 64-bit
Linux 32-bit
Linux ARMv6hf
Mac OS X

Processing è **disponibile per vari sistemi operativi**,
tra cui Windows, Linux e Mac OS X.
Scaricandolo avremo accesso all'ultima versione disponibile, la 3.3.6.
Successivamente averlo scaricato, basta scompattare il file e *avviarlo*
seguendo le corrette indicazioni. Ora potremo iniziare a utilizzarlo.

Prima versione: 1.0 il 29 novembre 2008

Sito originale: https://processing.org/

Caratteristiche:

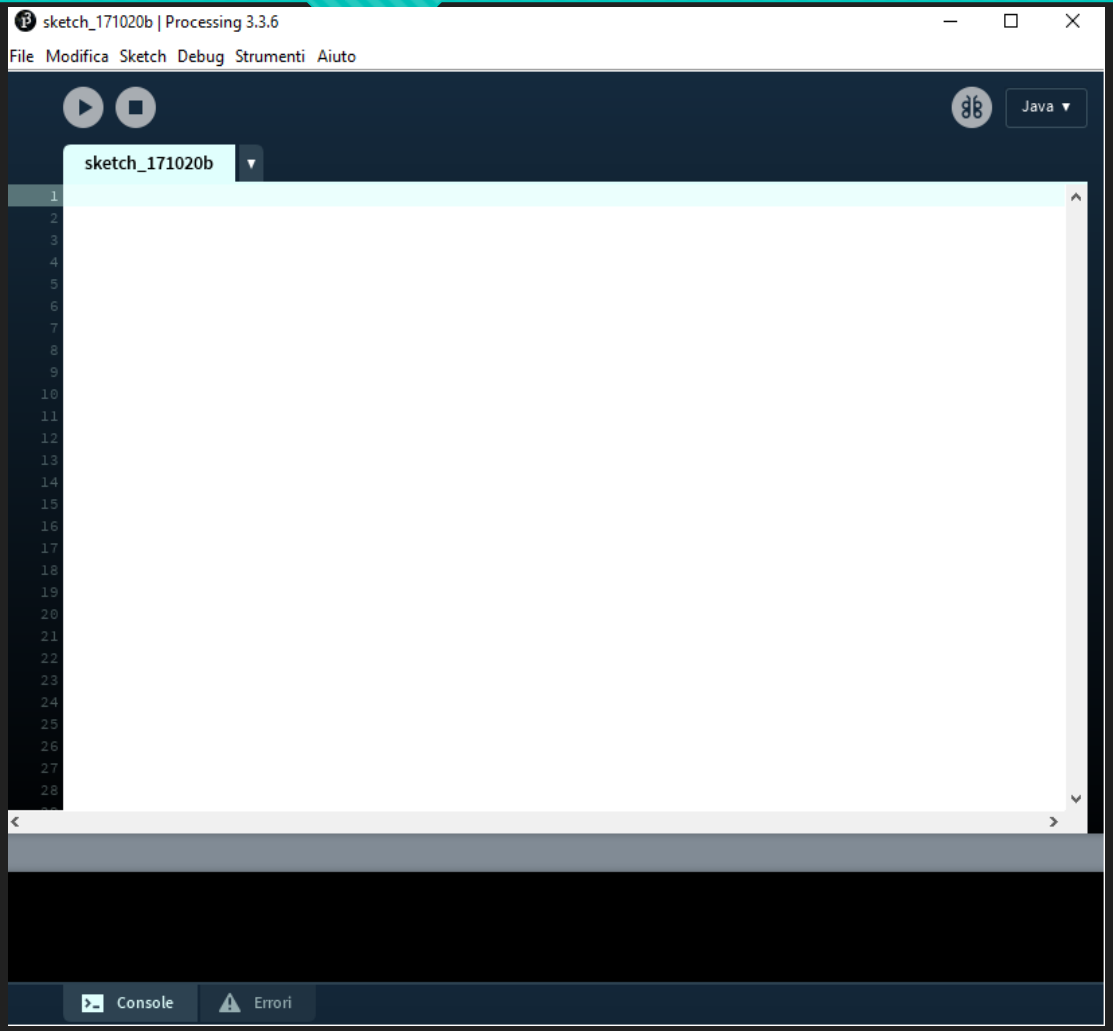
Processing eredita la maggior parte della sintassi da **Java** (*Java-Based*) ma in più mette a disposizione numerose **funzioni** ad alto livello per gestire in modo facile gli aspetti grafici e multimediali. Le varie **creazioni** sono chiamate ***«sketch»*** (*come su Arduino*) e sono organizzate in uno ***'sketch-book'***.

Ogni sketch contiene, in genere, oltre alle classi di oggetti che lo compongono, anche una cartella Data in cui viene inserito il **materiale multimediale** utile all'applicazione (ad esempio, immagini, font e file audio).

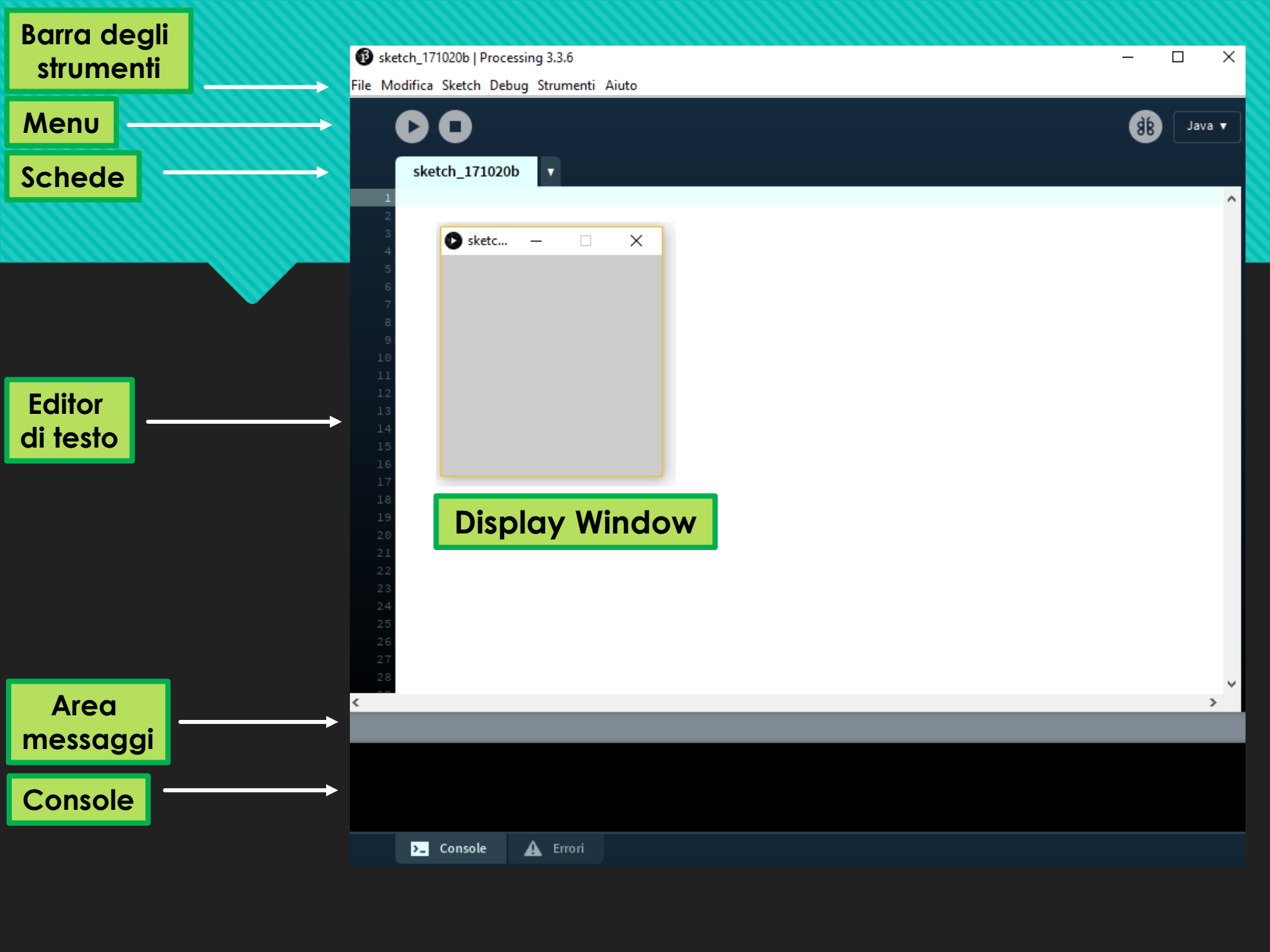
In Processing, ogni sketch può contenere almeno una funzione, tra cui setup e draw. La prima viene invocata una sola volta al lancio dell'applicazione, mentre la seconda viene eseguita per ogni frame.

Interfaccia:

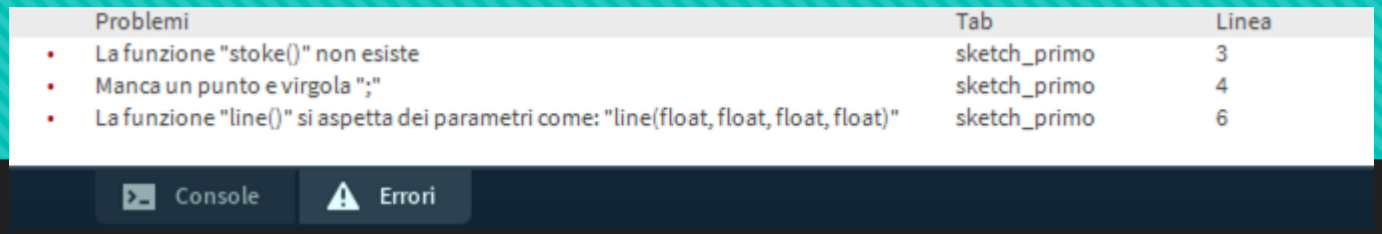
Prima di cominciare a scrivere le prime linee di codice, è importante capire quali sono le potenzialità di questo linguaggio. Se avete già installato il software sul vostro computer, attraverso il menu *File > Examples*, potete navigare in una serie di esempi suddivisi in varie categorie: *Basics, Topics, Demos, Books, Libraries* e *Contributed Libraries*.



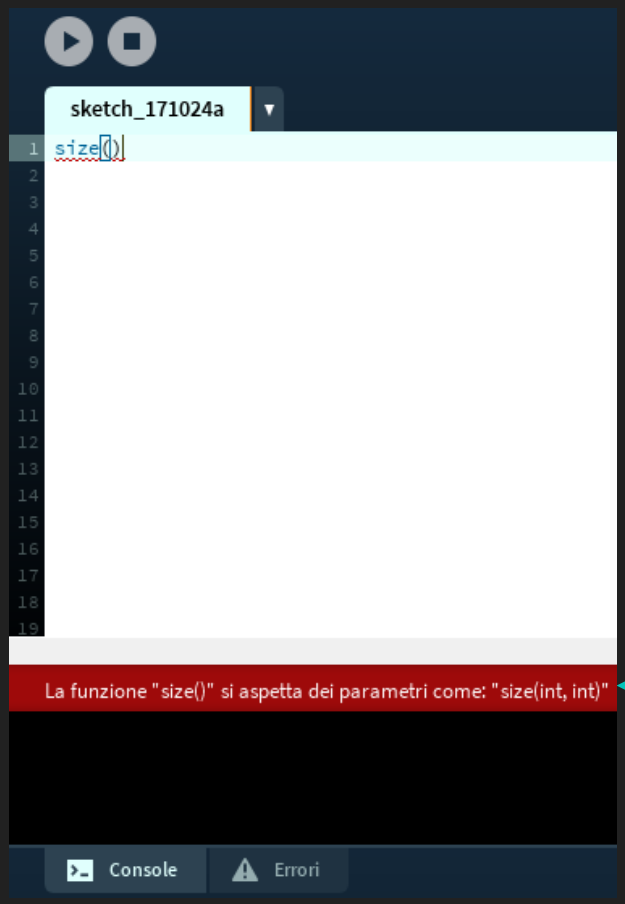
L'interfaccia è molto semplice: in alto troviamo i pulsanti **Run** e **Stop** che utilizzeremo per avviare e fermare i nostri ***sketch*** e, accanto a essi, i pulsanti per creare, aprire o salvare un nuovo progetto o per esportare l'applicazione. La parte centrale con sfondo bianco è il nostro editor mentre la sezione in basso con sfondo nero è la console di debugging.


Barra degli strumenti
Menu
Schede
Editor di testo
Area messaggi
Console
sketch_171020b | Processing 3.3.6
File Modifica Sketch Debug Strumenti Aiuto
Java
sketch_171020b
sketc...
Display Window
Console
Errori

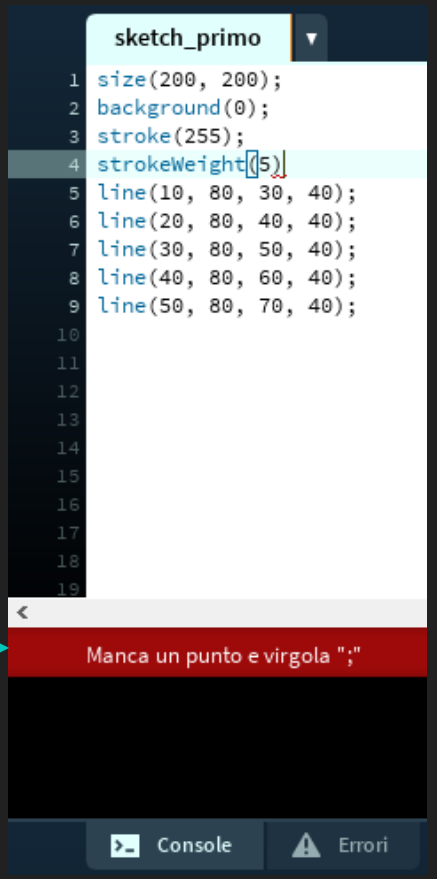
Processing è **tradotto completamente in italiano**, questo facilita la programmazione rendendo più **veloce la scrittura di programmi** e la rilevazione di eventuali **errori**.



Nella sezione errori vengono mostrati gli sbagli commessi lungo tutto il programma **specificando la linea e il tipo di inesattezza** (ed eventualmente la classe).



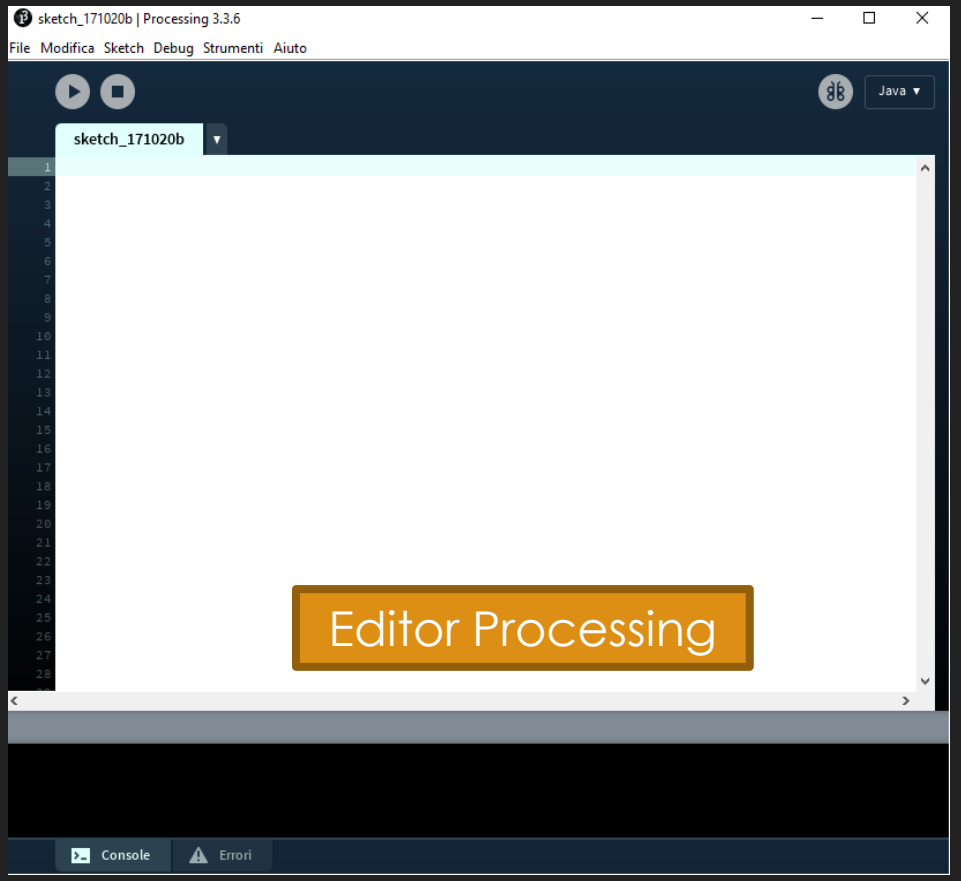
Quando si commettono sbagli nella linea in cui si sta digitando l'errore viene **segnalato immediatamente** nell'Area Messaggi.



Processing e Arduino:

Processing ricorda molto la nota applicazione «Arduino» (questo perché l'applicazione è una versione semplificata di Processing e modificata per Arduino), famosa perché consente di programmare l'omonima board. Processing è in grado di **interfacciarsi con Arduino**, questo permette di **controllare Arduino (anche a distanza)** e di **vedere il funzionamento del circuito su schermo**.

Editor Arduino



Editor Processing

Esempio programma semplice:

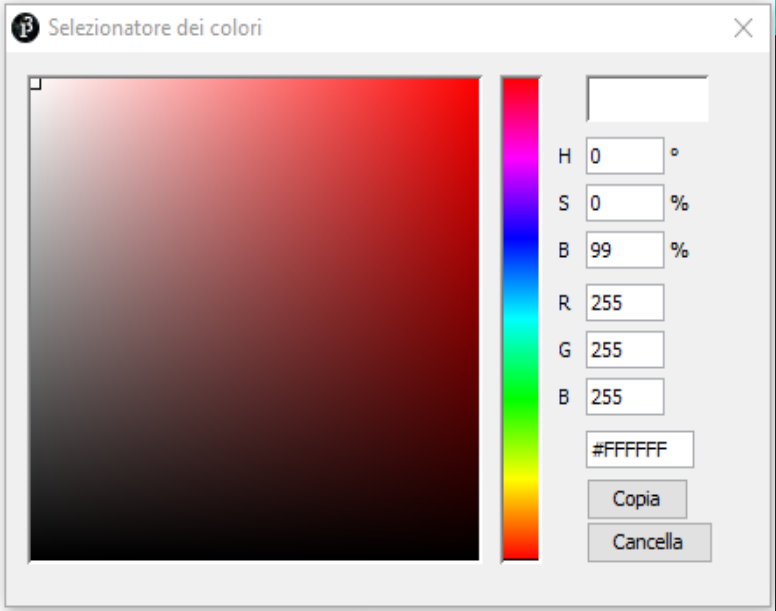
sketch_primo | Processing 3.3.6

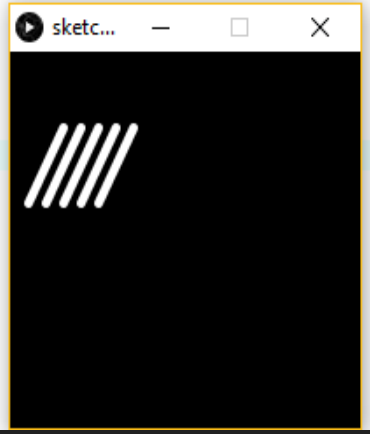
File Modifica Sketch Debug Strumenti Aiuto

sketch_primo

```
1  size(200, 200);
2
3  //Imposta la grandezza della finestra: il primo è la x il secondo la y
4
5  background(0);
6
7  //Imposta il colore dello sfondo, si può impostare:
8  //un colore (scala RGB, 3 numeri interi da 0 a 255 per le gradazioni,
9  //un colore in scala grigi (un numero intero per le gradazioni),
10 //un immagine (bisogna prima scaricarla o impostare l'url)
11
12 stroke(255);
13
14 //Imposta il colore di ciò che vogliamo disegnare (inquesto caso 255 è il bianco)
15
16 strokeWeight(5);
17 //Imposta la grandezza in pixel di ciò che si vuole disegnare
18 line(10, 80, 30, 40);
19 //Disegna una linea che parte da (x = 10, y = 80) e termina in (x = 30, y = 40)
20 line(20, 80, 40, 40);
21 //Disegna una linea che parte da (x = 20, y = 80) e termina in (x = 40, y = 40)
22 line(30, 80, 50, 40);
23 //Disegna una linea che parte da (x = 30, y = 80) e termina in (x = 50, y = 40)
24 line(40, 80, 60, 40);
25 //Disegna una linea che parte da (x = 40, y = 80) e termina in (x = 60, y = 40)
26 line(50, 80, 70, 40);
27 //Disegna una linea che parte da (x = 50, y = 80) e termina in (x = 70, y = 40)
28
29
30
31
32
33
34
35
36
```

Codice del programma





Risultato dello sketch

Le **forme base** della **geometria** sono disegnate con processing attraverso delle funzioni:

Le funzioni fondamentali sono:

Il punto

La funzione ***point()*** disegna un punto di dimensione pari a 1 pixel sullo schermo. Accetta due parametri: **x** e **y** che sono, rispettivamente la coordinata orizzontale e quella verticale.

La linea

Per disegnare una linea (funzione ***line()***) abbiamo bisogno di quattro parametri: **x** e **y** del punto di partenza e di arrivo.

Il rettangolo

La funzione ***rect()*** ha bisogno di quattro parametri: la **x** e la **y** del punto di origine, e poi larghezza e l'altezza del rettangolo. Per disegnare un quadrato basta impostare la larghezza e l'altezza uguali.

L'ellisse

Questa funzione, ***ellipse()***, accetta quattro parametri: **x** e **y** del punto di origine più la larghezza e l'altezza dell'ellisse. Per disegnare un cerchio si impostano le ultime due uguali.

Il triangolo

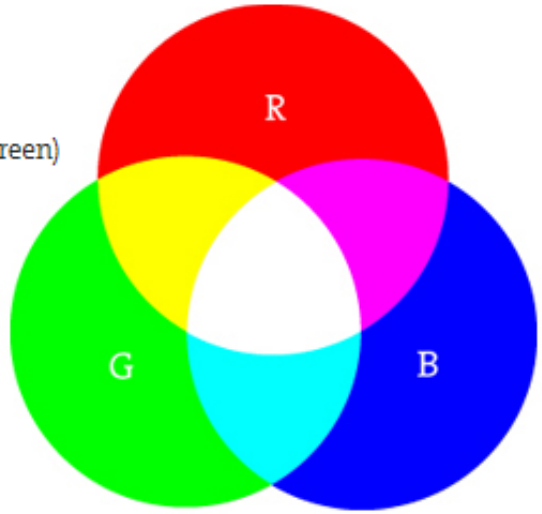
Grazie alla funzione ***triangle()*** possiamo disegnare un triangolo indicando come parametri le coordinate **x** e **y** dei tre vertici del triangolo.

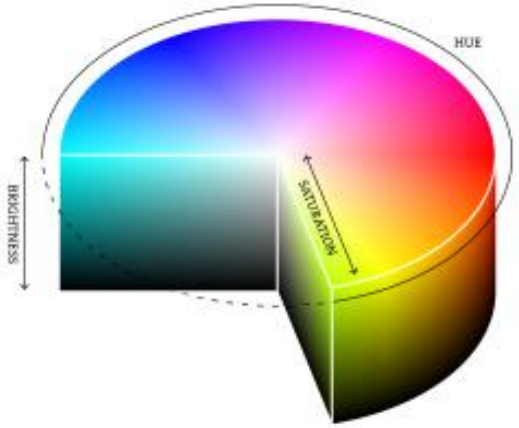
I colori:

RGB: **Red**, **Green** and **Blue**:

Prima di cominciare a utilizzare i colori è importante capire come funzionano. Abbiamo tutti imparato che attraverso la miscelazione dei tre colori primari, **rosso**, **giallo** e **blu**, si possono ottenere tutti i colori nelle diverse sfumature. Anche gli schermi funzionano in modo simile ma i tre colori di base sono **rosso**, **verde** e **blu** in inglese **RGB**.

- Red + Green = Yellow
- Red + Blue = Purple
- Green + Blue = Cyan (blue-green)
- Red + Green + Blue = White
- No colors = Black





- **Hue**—The color type, ranges from 0 to 255 by default.
- **Saturation**—The vibrancy of the color, 0 to 255 by default.
- **Brightness**—The, well, brightness of the color, 0 to 255 by default.

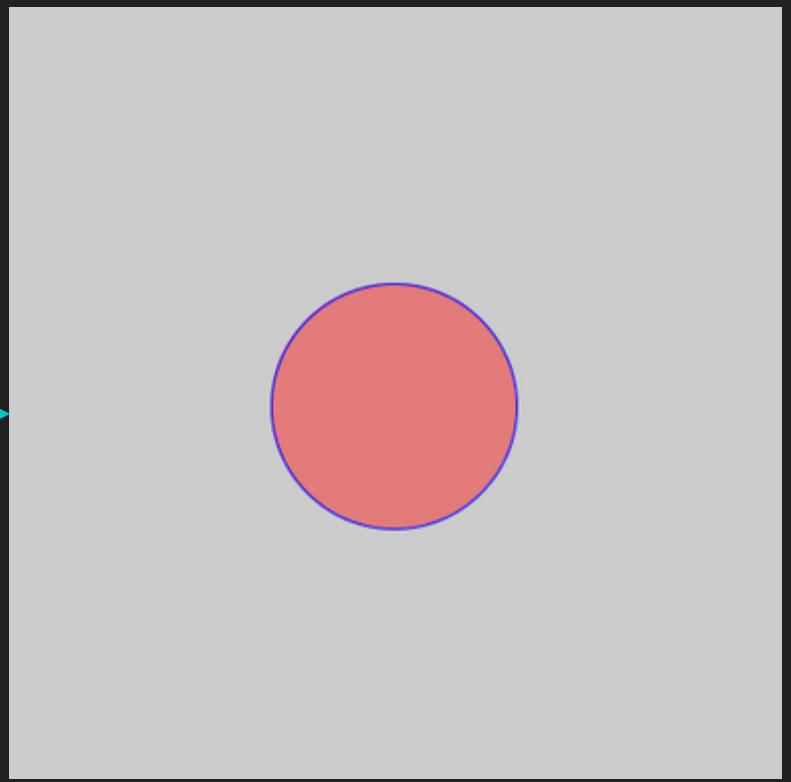
Colori in scala di grigi, dove il nero ha valore 0 e il bianco 255.

In Processing esistono due tipi di funzioni per colorare le forme: **fill()** e **stroke()**. Con la prima indichiamo il colore di riempimento della forma, mentre con la seconda, il colore del bordo.

sketch_171102a

```
1 size(500, 500);
2 fill(255, 0, 0, 100);
3 stroke(0, 0, 255);
4 ellipse(250, 250, 150, 150);
```

Se si vuole modificare anche l'opacità del colore basta **aggiungere un parametro** (un numero da 0 a 255) che rappresenta appunta la trasparenza



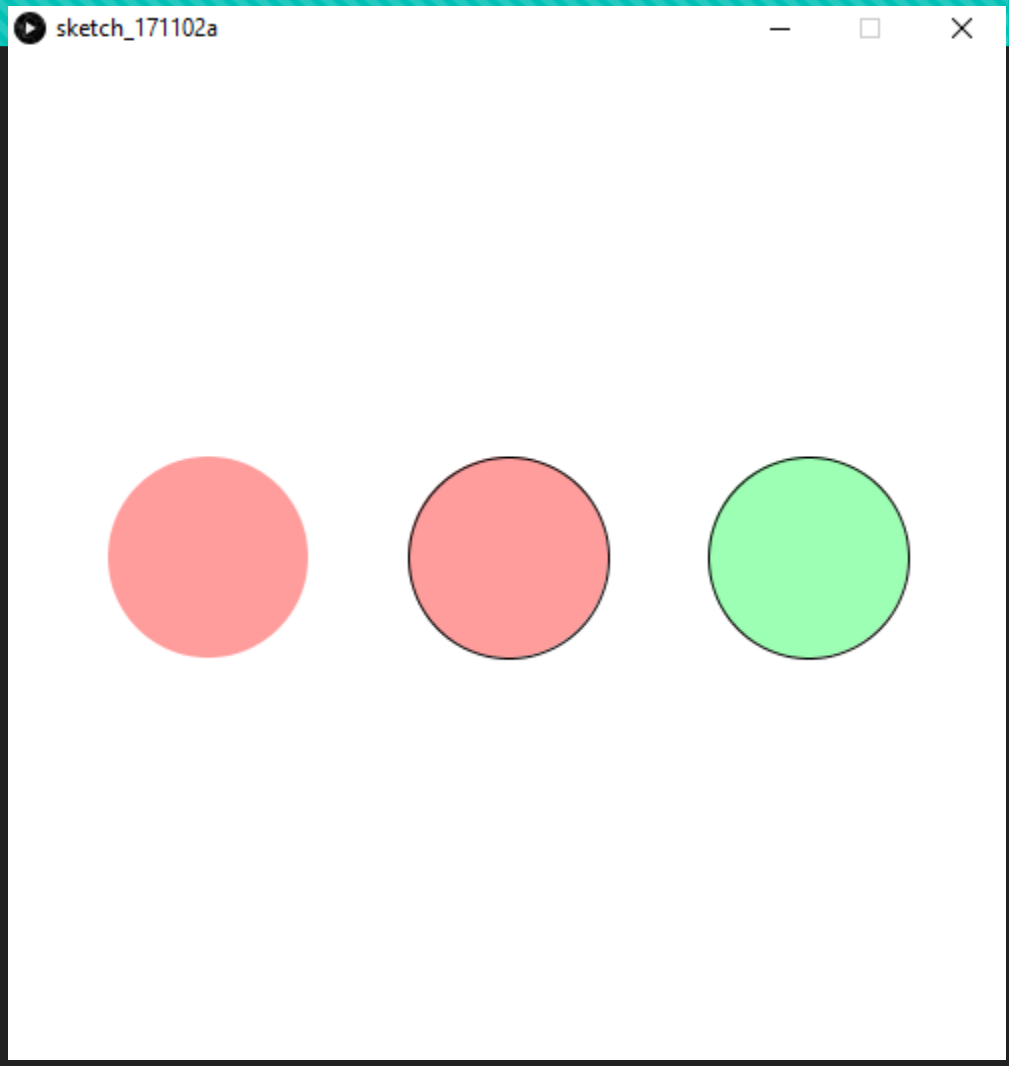
Ecco un esempio per capire meglio le funzioni dei colori:

sketch_171102a

```
1  // Impostiamo la dimensione della finestra a 500x500px
2  size(500, 500);
3  // Impostiamo lo sfondo di colore bianco
4  background(255);
5  // Impostiamo di non avere un bordo
6  noStroke();
7  // Impostiamo una tonalità di rosso come riempimento
8  fill(255, 157, 157);
9  // Disegnamo il primo cerchio
10 ellipse(100, 250, 100, 100);
11 // Impostiamo il bordo di colore nero
12 stroke(0);
13 // Disegniamo il secondo cerchio
14 ellipse(250, 250, 100, 100);
15 // Impostiamo una tonalità di verde come riempimento
16 fill(157, 255, 179);
17 // Disegniamo il terzo cerchio
18 ellipse(400, 250, 100, 100);
```

Le funzioni **noFill** e **noStroke** servono per determinare un **nessun riempimento** o un **nessun contorno** delle figure che si andranno a disegnare

Esempio programma semplice:



Come abbiamo già accennato, un programma in Processing è costituito da **due funzioni principali**, appunto **setup()** e **draw()**.

L'inserimento di queste funzioni nel programma normalmente **non è obbligatorio**. Infatti si possono scrivere programmi anche senza usare una di queste due funzioni o anche senza usare ***nessuna funzione***.

Diventa obbligatorio metterle quando si vogliono creare animazioni (ad esempio video animati, giochi…)

La funzione setup() è la prima ad essere eseguita nel programma. Una volta terminate le sue istruzioni, il programma passa alla funzione draw(), continuando a ripeterla poi all'infinito.

La velocità con cui si svolgono le istruzioni nella funzione draw() (quindi la velocità con cui si aggiornano le immagini) sono impostate a 60 **fps** (frame per second), misura modificabile attraverso la funzione.

Oltre ai tipi di variabili, Processing possiede delle sue variabili (definite **variabili built-in**) che consentono di agevolare la scrittura di un programma.

Processing, come già accennato, deriva dal linguaggio di programmazione *Java*, per questo **eredita** da lui anche **tutte le sue caratteristiche**.

Infatti in Processing si possono creare tutte le variabili che si possono creare in Java:

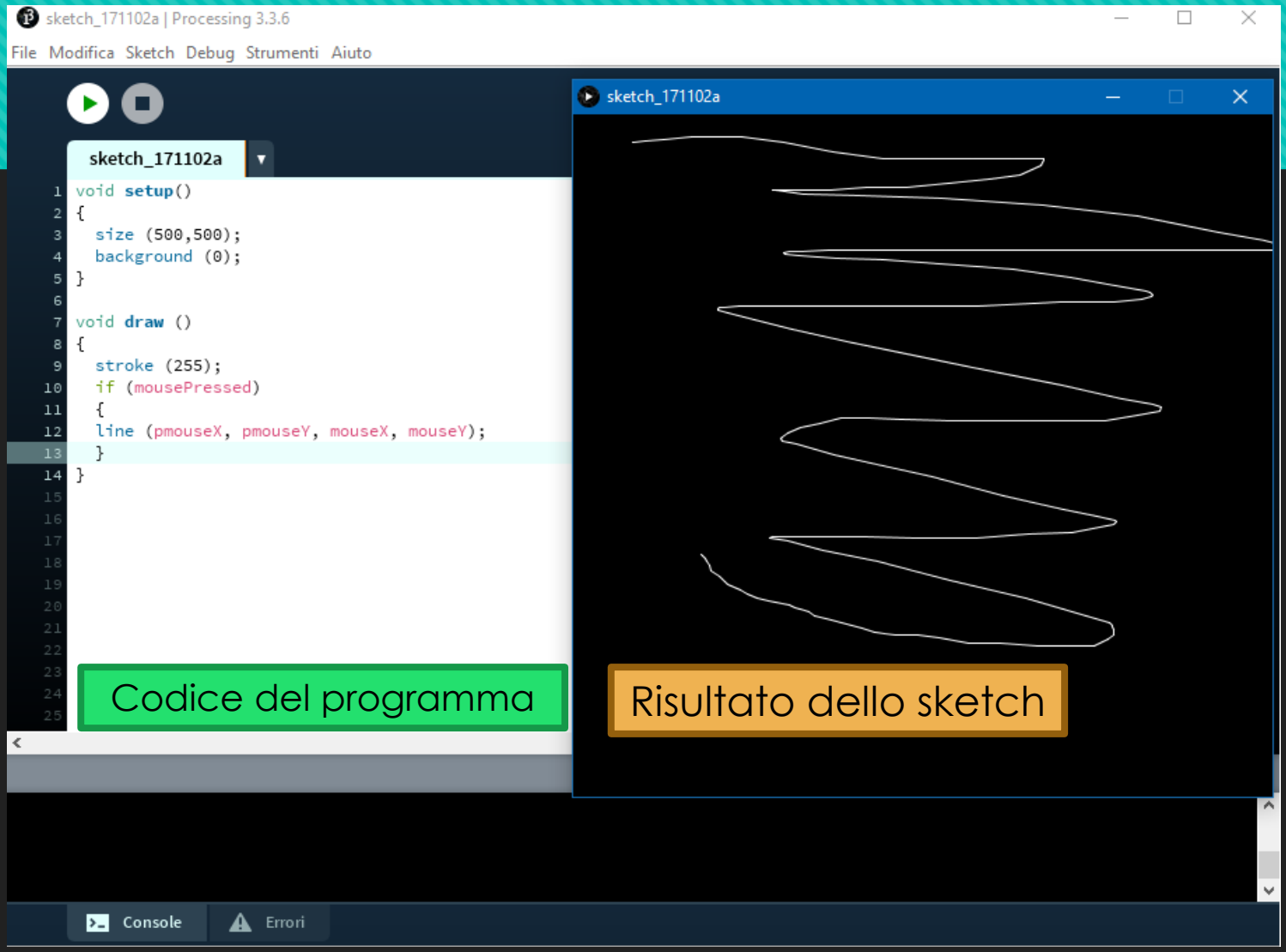
→ Ad esempio: *int*, *float*, *boolean*, *char*…

Variabili ***mouseX*** e ***mouseY***:
se inseriamo queste variabili, all'interno del blocco di codice *draw()* ci restituiscono in tempo reale i valori ***X*** e ***Y del mouse*** quando ci muoviamo all'interno della finestra che abbiamo creato.

→ Varabili ***pmouseX*** e ***pmouseY***: a differenza di mouseX e mouseY, queste variabili restituiscono i valori x e y del mouse del frame *precedente* a quello corrente.

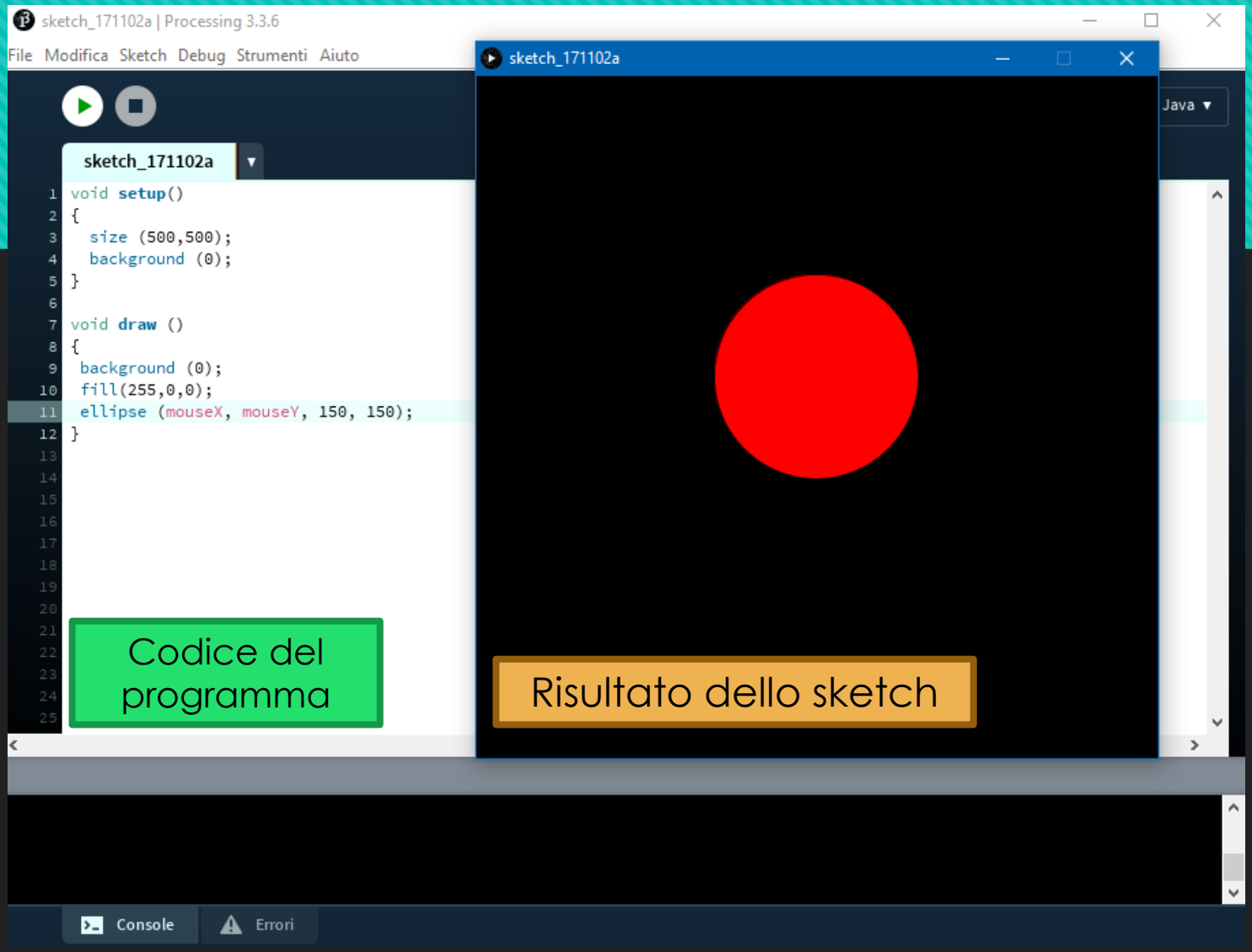
Width e ***height***:
Width e *height* sono variabili che restituiscono le dimensioni della finestra dopo che queste sono state impostate nel blocco di codice *setup()*.

Ecco un piccolo esempio delle funzioni inerenti al **cursore del mouse**:



Con questo sketch abbiamo creato un «foglio di lavoro» su cui possiamo liberamente disegnare tenendo premuto il mouse

Un secondo esempio semplice, la nostra prima **animazione**:



Con questo sketch invece abbiamo creato un cerchio rosso che segue le coordinate del nostro puntatore del mouse

Comportandosi in modo simile a ***setup()*** e ***draw()***, **gli eventi** sono descritti anch'essi, come blocchi di codice. La loro particolarità è che il codice scritto al loro interno **non viene eseguito automaticamente all'avvio del programma** ma al verificarsi di una condizione specifica come, per l'appunto, il click del mouse. A differenza della funzione **draw()**, le porzioni di codice all'interno degli eventi non vengono eseguiti in **loop** ma una volta soltanto.

Gli eventi del mouse:

Gli eventi del mouse sono gestiti da funzioni e da variabili built-in e sono:

mouseButton, mouseClicked(), mouseDragged(), mouseMoved(), mousePressed(), mousePressed, mouseReleased(), mouseWheel().

Gli eventi della tastiera:

Gli eventi della tastiera sono gestiti da funzioni e da variabili built-in e sono:

keyPressed(), keyReleased(), keyPressed, key.

Esempio programma che gestisce gli eventi del mouse:

Quando il programma riconosce la pressione di un tasto sul mouse (basta un click, che sia del tasto sinistro o destro è indifferente), il colore di sfondo varia dal bianco al rosso, quando il mouse invece è rilasciato il colore in background torna ad essere bianco.

Codice del programma

Sketch quando il mouse non è premuto

Sketch quando il mouse è premuto

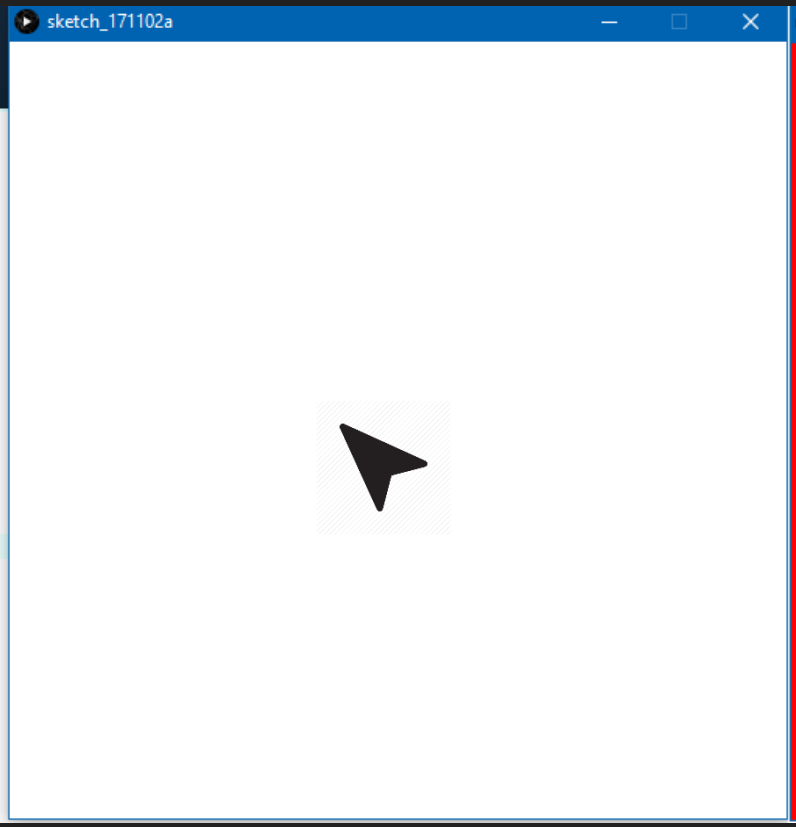
sketch_171102a

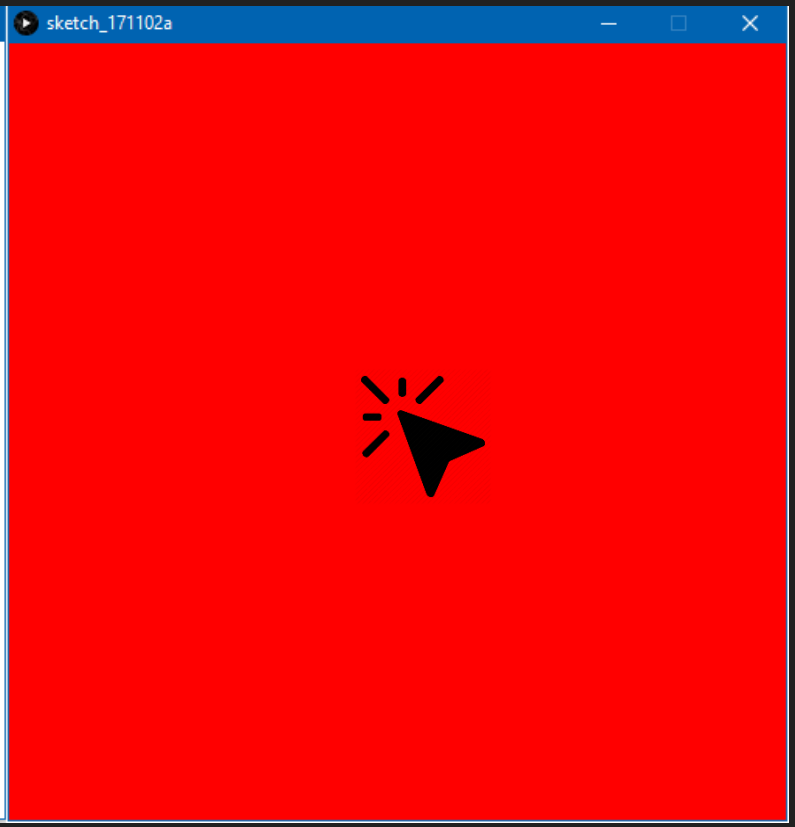
```
void draw() {
}

void keyPressed() {
  background(255, 0, 0);
}

void keyReleased() {
  background(255);
}

void mousePressed() {
  background(255, 0, 0);
}
void mouseReleased() {
  background(255);
}
```





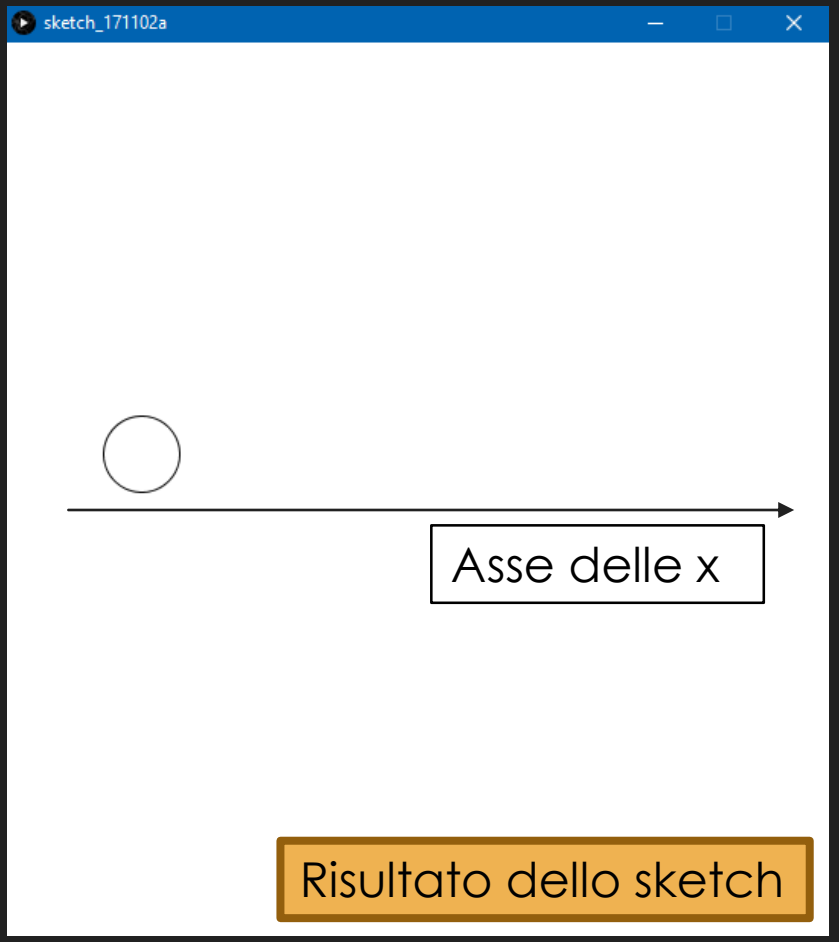
Le **variabili sono molto importanti** anche *per le animazioni*, ad esempio ci permettono di cambiare posizioni ad oggetti, di eseguire delle azioni complesse in un determinato momento.

sketch_171102a

```
1  int ellipseX; // dichiarazione della variabile
2
3  void setup() {
4    size(540, 540);
5    ellipseX = 0; // inizializzazione
6
7  }
8
9  void draw() {
10   background(255);
11   ellipse(ellipseX, height/2, 50, 50);
12   ellipseX = ellipseX + 1; // aumento il valore di ellipseX
13   println(ellipseX);
14 }
```

Codice del programma

Ad esempio, questa è l'animazione di un cerchio che si muove lungo l'asse delle x



Risultato dello sketch

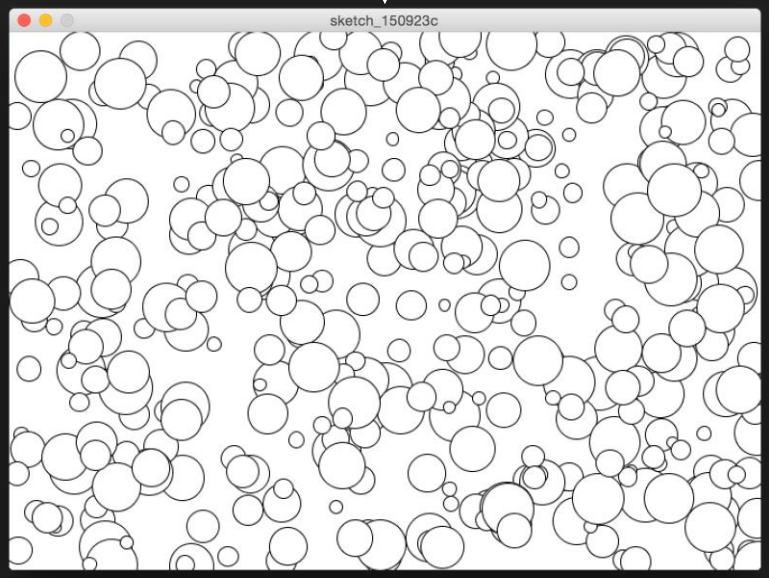
Il cerchio segue l'asse X per il suo spostamento orizzontale

Un altro esempio è riportato nella figura sottostante. In quest'ultima è stata aggiunta una funzione, **random()**, *che genera* ***numeri casuali***. In essa si specificano **due termini**: il valore iniziale e quello finale tra i quali estrarre il numero, oppure una variabile che rappresenta il **numero massimo** che la funzione può restituire.

disegno_sfondo_eventi

```
1  float radius;
2
3  void setup() {
4    size(700, 500);
5    background(255);
6  }
7  void draw() {
8    radius = random(10, 50);
9    ellipse(random(width), random(height), radius, radius);
10 }
```

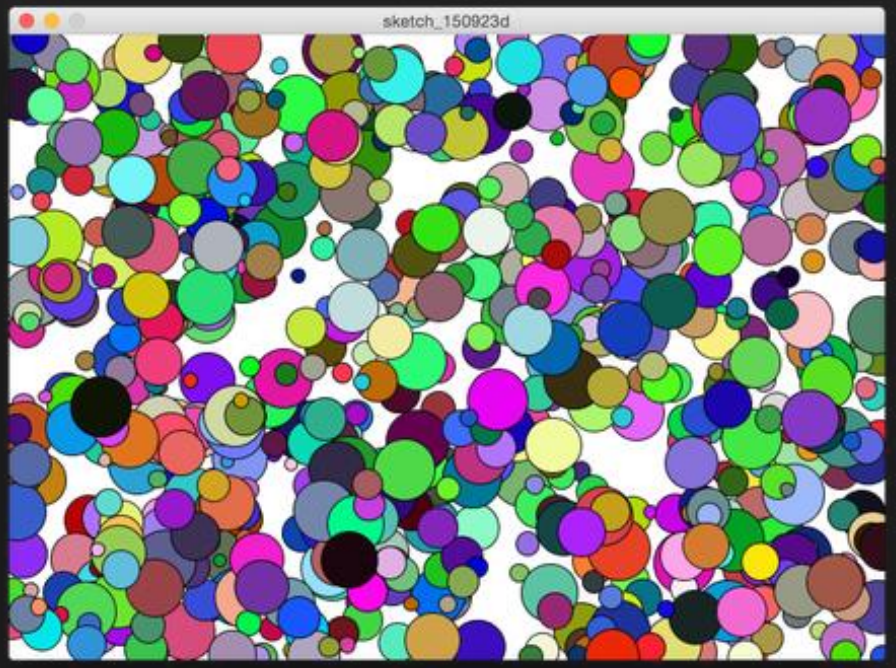
Codice del programma



Se aggiungiamo la riga di istruzione seguente:

fill(random(255), random(255), random(255));

Lo sketch **cambia** in questo modo:



In Processing valgono inoltre tutte le strutture che conosciamo dal codice di programmazione C++ con l'aggiunta di nuovi particolari.

Strutture di Processing:

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

If
Else
While
For
Do - While
Else - If

L'Else - If discende dal linguaggio Java.
In poche parole consente di creare **«If»** consecutivi. Quando poniamo una qualsiasi condizione, sotto di essa possiamo aggiungere infinite funzioni **Else - If**, sotto alle quali, al termine, possiamo chiudere con la funzione else.

In questo esempio vediamo come **lavora la funzione Else-If**:

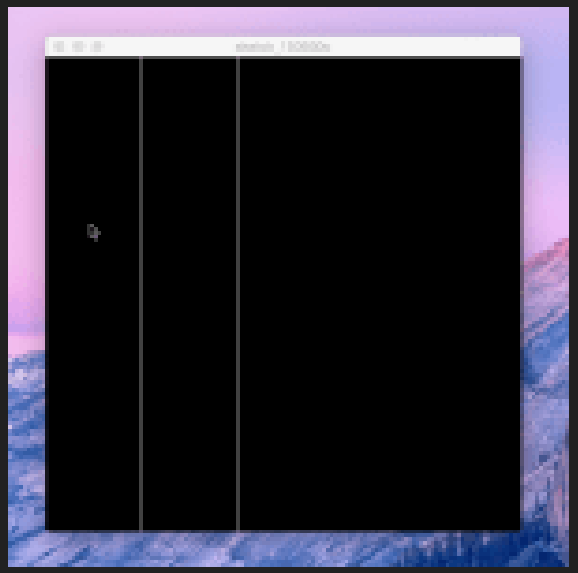
strutture_condizionali

```
1  void setup() {
2    size(500, 500);
3    stroke(255);
4  }
5  void draw() {
6    if (mouseX > 200) {
7      background(200);
8    } else if (mouseX > 100) {
9      background(127);
10   } else {
11     background(0);
12   }
13   line(100, 0, 100, height);
14   line(200, 0, 200, height);
15 }
```

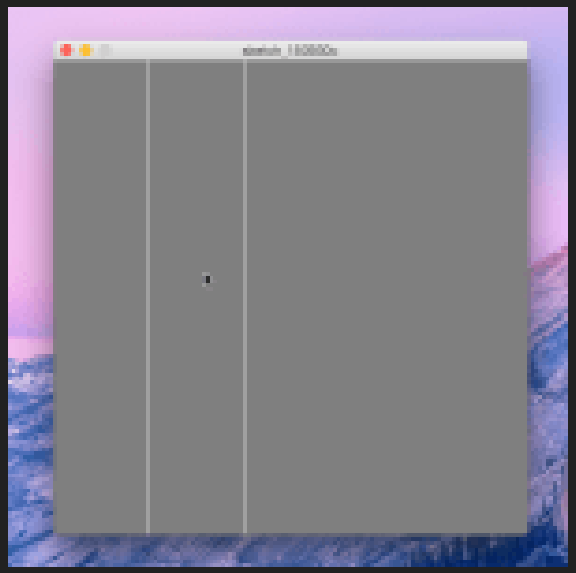
Codice del programma

Questo programma mostra l'utilità della funzione Else-If. Quando il puntatore del mouse è posizionato sul primo rettangolo, il colore di sfondo è nero. Quando invece lo spostiamo, il colore di sfondo varia in una tonalità di grigio. Infine se lo portiamo sull'ultimo parallelepipedo, lo sfondo di background appare quasi bianco.

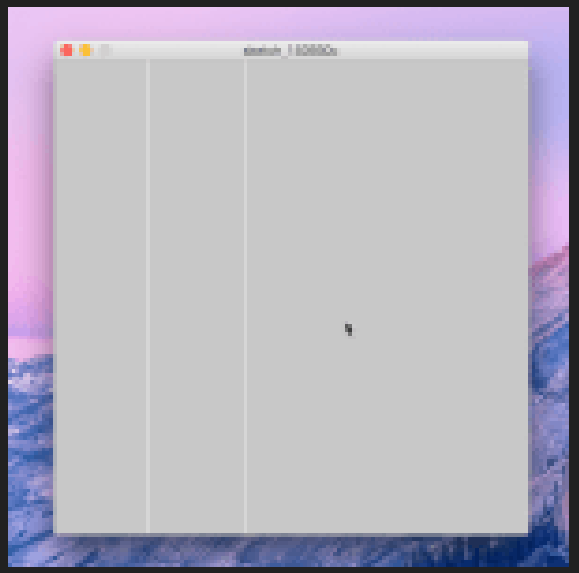
PRIMO CASO:



SECONDO CASO:



TERZO CASO:



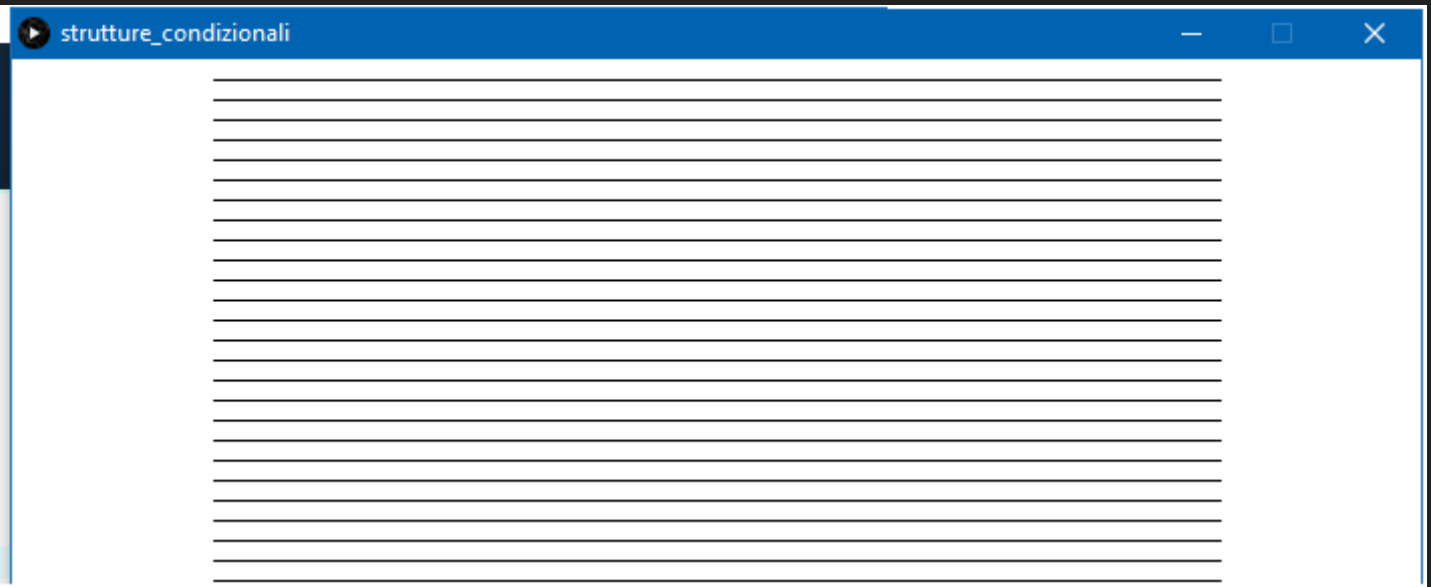
In questo esempio possiamo osservare l'uso del **ciclo while**.

In queste semplici righe di comando, è spiegato il **ciclo while**. Questo programma ripete delle linee continue, distanziate tra di loro di 10 pixel. Il tutto è disegnato su uno sfondo bianco.

Nel loop while, il blocco di codice verrà eseguito **finché** la condizione inserita tra le parentesi è **vera**.

strutture_condizionali

```
1  int y = 10;
2
3  void setup() {
4    size(700, 500);
5    background(255);
6    while(y < height) {
7      line(100, y, 600, y);
8      y += 10;
9    }
10 }
11 void draw() {
12 }
```



Codice del programma

Risultato dello sketch

Un atro esempio con altre strutture:

Questo programma costruisce un quadrato suddiviso in 4 sezioni. La prima in alto a sinistra (se viene passato il puntatore del mouse sopra) lo sfondo diventa rosso. La seconda sezione diventa verde, e le ultime due restanti tengono il background nero.

Codice del programma

PRIMO CASO:

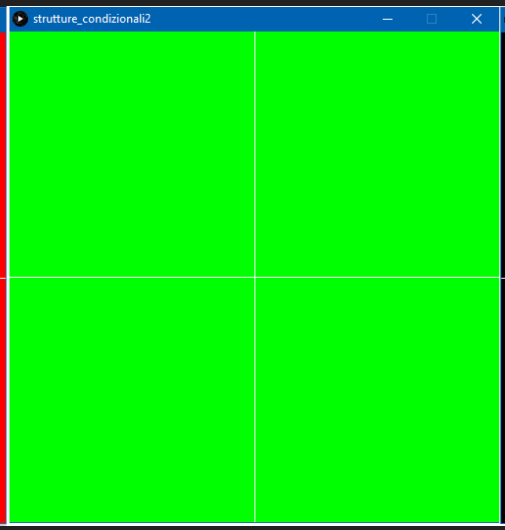
SECONDO CASO:

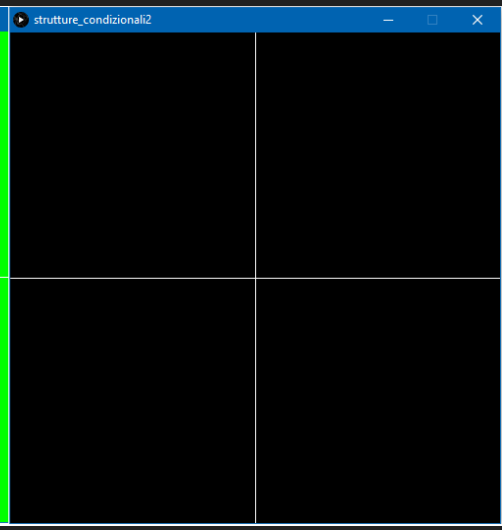
TERZO CASO:

strutture_condizionali2

```
void setup() {
  size(500, 500);
  stroke(255);
}

void draw() {
  if((mouseX < width/2) && (mouseY < height/2)) {
    background(255, 0, 0);
  } else if((mouseX > width/2) && (mouseY < height/2)) {
    background(0, 255, 0);
  } else {
    background(0);
  }
  line(width/2, 0, width/2, height);
  line(0, height/2, width, height/2);
}
```





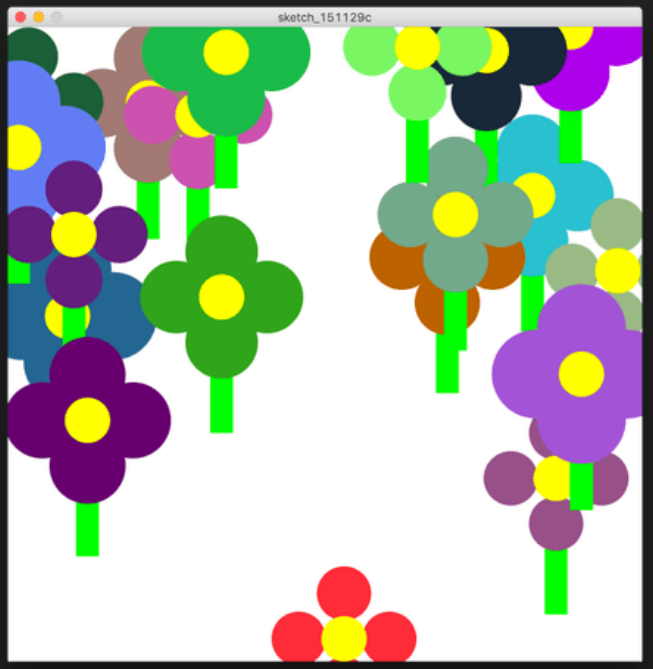
Il codice di questo programma si chiama **funzioni1**. In questo si nota che la funzione **flower** consente di creare un fiore.

funzioni1

```
1  void setup() {
2    size(700, 700);
3    background(255);
4  }
5
6  void draw() {
7    flower(width/2, height/2);
8  }
9
10 void flower(int posizioneX, int posizioneY) {
11   noStroke();
12   fill(0, 255, 0);
13   rectMode(CENTER);
14   rect(posizioneX, posizioneY+100, 25, 100);
15   fill(255, 0, 255);
16   ellipse(posizioneX-50, posizioneY, 70, 70);
17   ellipse(posizioneX, posizioneY-50, 70, 70);
18   ellipse(posizioneX+50, posizioneY, 70, 70);
19   ellipse(posizioneX, posizioneY+50, 70, 70);
20   fill(255, 255, 0);
21   ellipse(posizioneX, posizioneY, 50, 50);
22   noFill();
23 }
```

SKETCH

Con un ciclo for continuiamo a creare dei fiori con la funzione flower **in posti casuali** (random), e in più i colori dei fiori sono casuali.

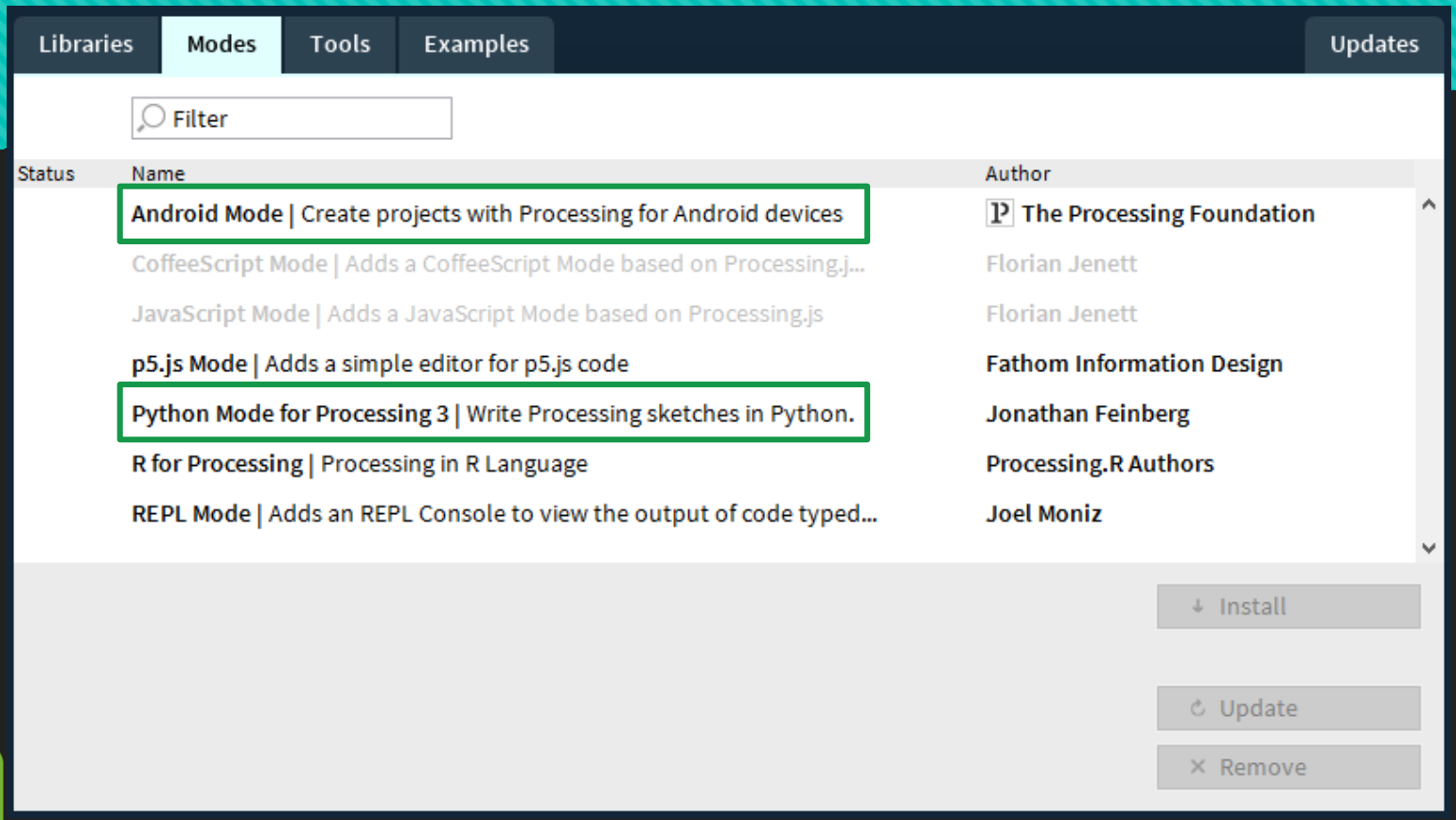


Nel programma di prima la funzione flower non ha molto senso perché viene usata una sola volta. In quest'altro esempio possiamo capirne l'utilità.

Grazie a Processing è possibile **programmare** in parecchi **diversi linguaggi** e su molte piattaforme, ad esempio si possono creare applicazioni grafiche per Android OS, o programmare sketch in Python.

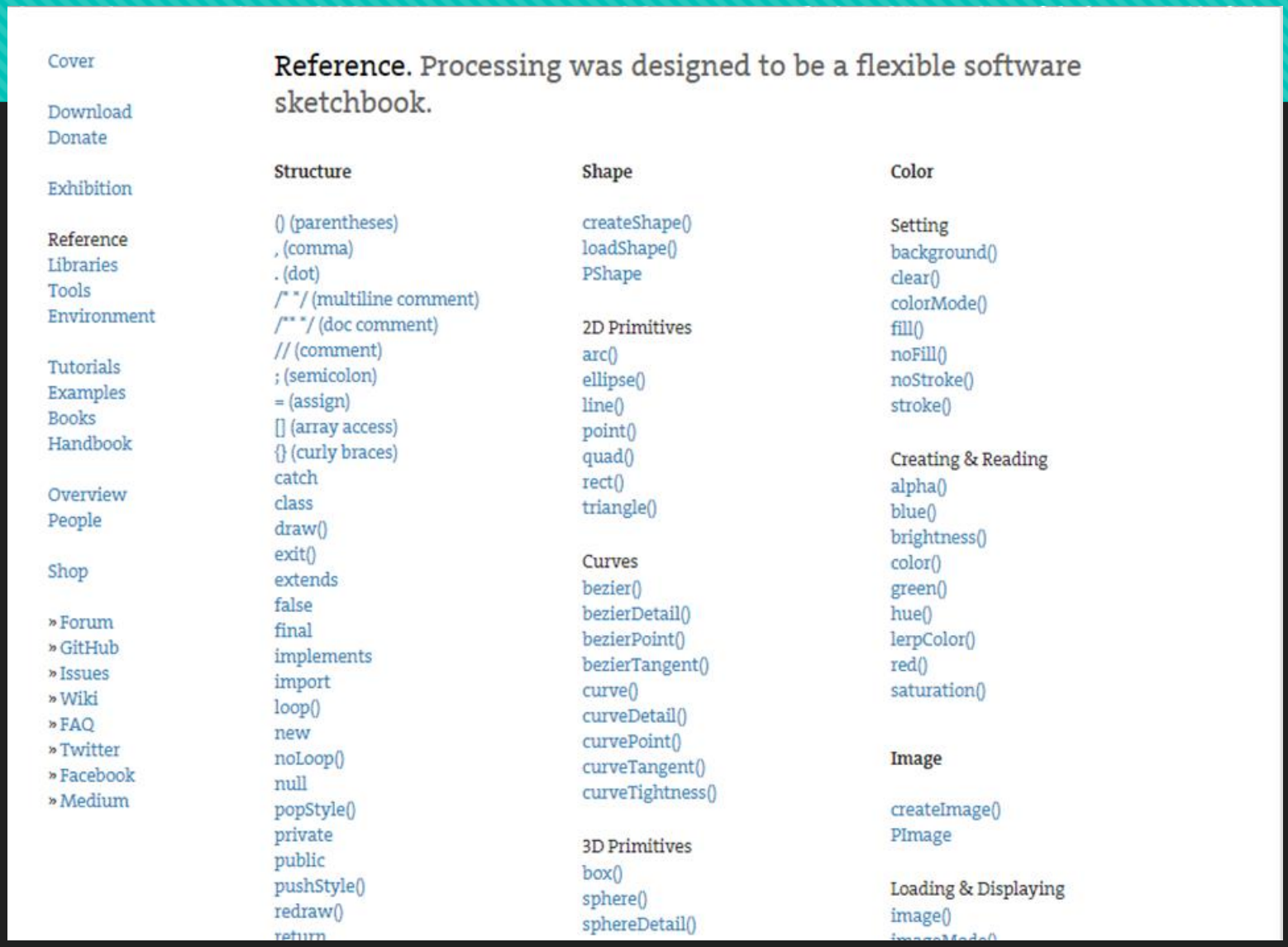
Android OS

Python



python™

Sul sito di Processing è presente una sezione **«*reference*»**, ovvero **«riferimento»** (in informatica si chiamano API, ovvero Application Programming Interface). Qui si possono trovare tutti i comandi di Processing e per ognuno di essi si può imparare come funziona, cosa fa, e molte altre informazioni a riguardo.



Cover

Download
Donate

Exhibition

Reference
Libraries
Tools
Environment

Tutorials
Examples
Books
Handbook

Overview
People

Shop

» Forum
» GitHub
» Issues
» Wiki
» FAQ
» Twitter
» Facebook
» Medium

Reference. Processing was designed to be a flexible software sketchbook.

Structure

() (parentheses)
, (comma)
. (dot)
/* */ (multiline comment)
/** */ (doc comment)
// (comment)
; (semicolon)
= (assign)
[] (array access)
{} (curly braces)
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
popStyle()
private
public
pushStyle()
redraw()
return

Shape

createShape()
loadShape()
PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()
sphere()
sphereDetail()

Color

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

Image

createImage()
PImage

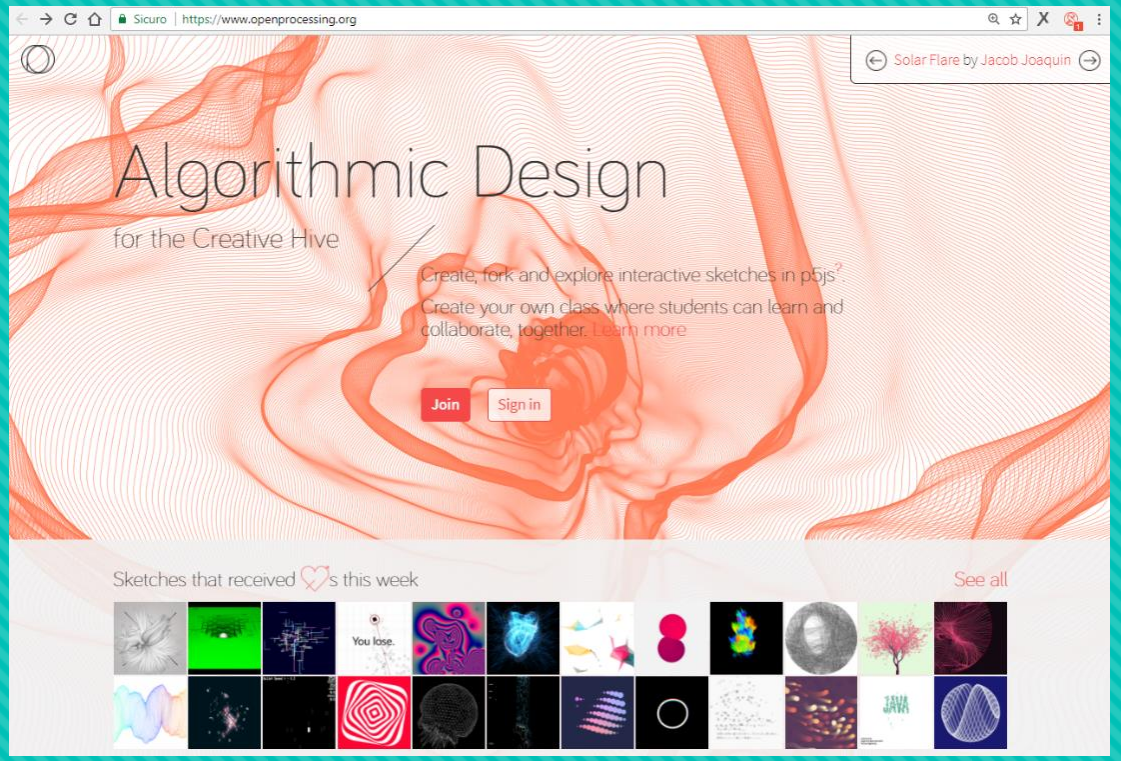
Loading & Displaying
image()

COMANDI

Processing 3

An open project initiated by Ben Fry and Casey Reas.
Supported by programmers like you and the nonprofit Processing Foundation, 501(c)(3).

https://www.openprocessing.org/



Ricordo che il pacchetto Processing v3.3.6 è scaricabile dal sito ufficiale gratuitamente.

Grazie per l'attenzione.