PROCESSING

Workshop intensivo Processing e Arduino

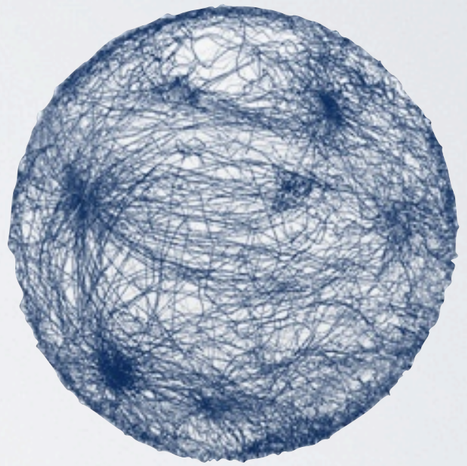
PROCESSING

E' un *linguaggio di programmazione* basato su Java che consente di sviluppare diverse applicazioni come giochi, animazioni e contenuti interattivi.

É distribuito sotto licenza Open Source ed è supportato dai sistemi operativi GNU/Linux, Mac OS X e Windows.

Realizzazione di Davide Cocchi
www.davidecocchi.com

Telekom Product Experience Center
Realtime Information Graphics
http://projects.zumkuckuck.com/realtime/



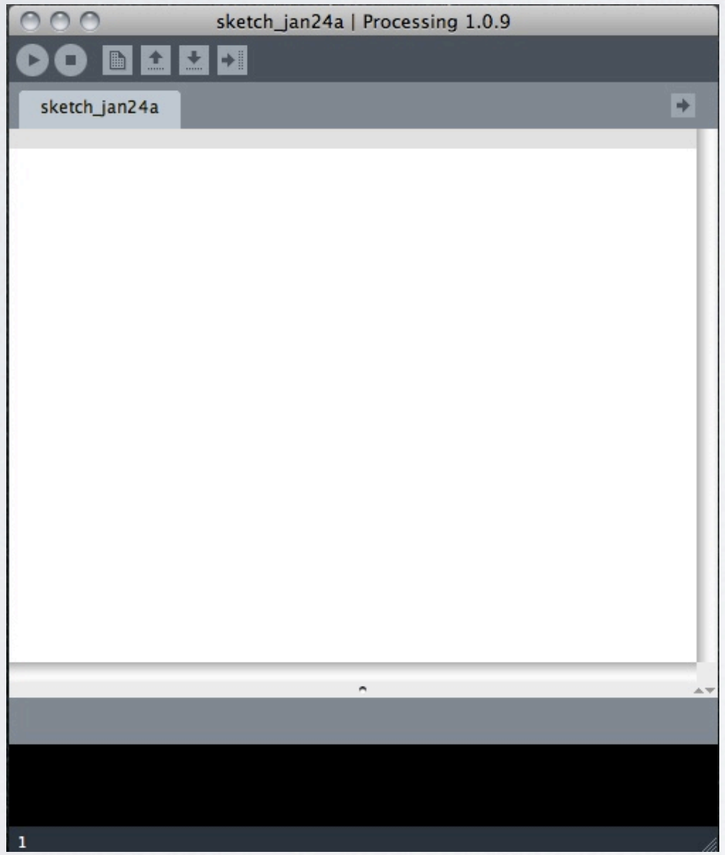
Generative identity software for COP15 the United Nations Climate Change Conference held in Copenhagen in 2009. http://www.okdeluxe.co.uk/cop15/

OPEN SOURCE

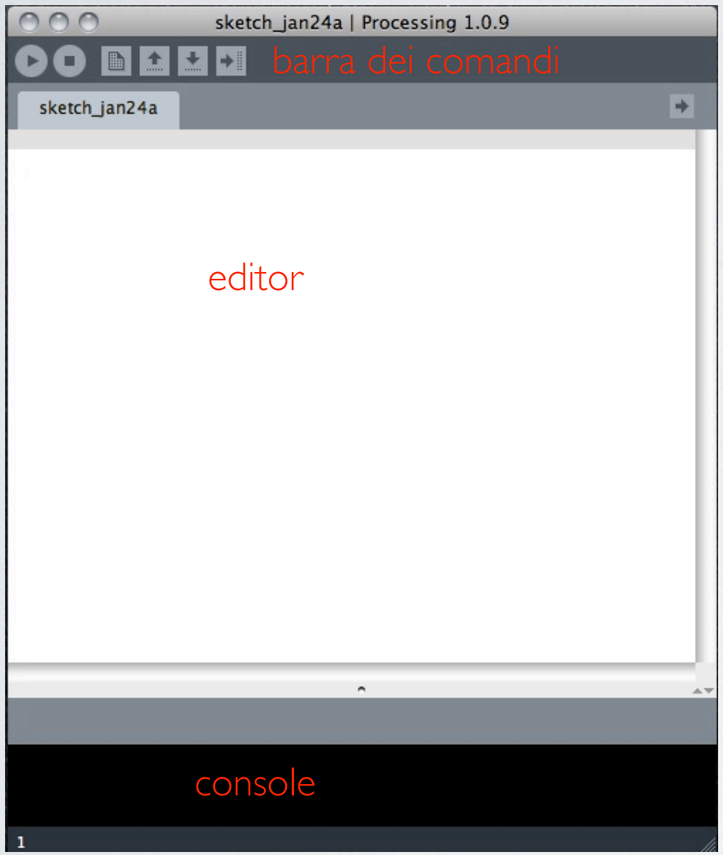
In informatica, **open source** (termine inglese che significa *sorgente aperto*) indica un software i cui autori (più precisamente i detentori dei diritti) ne favoriscono il libero studio e l'apporto di modifiche da parte di altri programmatori indipendenti. Questo è realizzato mediante l'applicazione di apposite licenze d'uso.

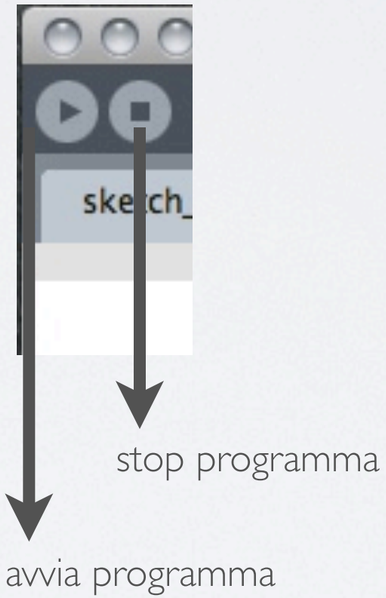
PROCESSING PUNTI FORTI

Semplicità ed **immediatezza** del codice, oltre a ciò, assieme al linguaggio si può scaricare un **IDE**, che permetterà di progettare senza dover ricorrere all'uso di ulteriori applicazioni di sviluppo.



PROCESSING IDE





PROCESSING PROGRAMMARE

Tre i metodi per programmare in Processing:

basic | intermediate | complex

Basic,che consiste in una sequenza di comandi per il disegno di primitive grafiche;

Intermediate, che è una programmazione procedurale (consiste nel creare dei blocchi di codice, identificati da un nome e racchiusi da dei delimitatori)

Complex, che consiste in una programmazione prevalentemente orientata agli oggetti Java.

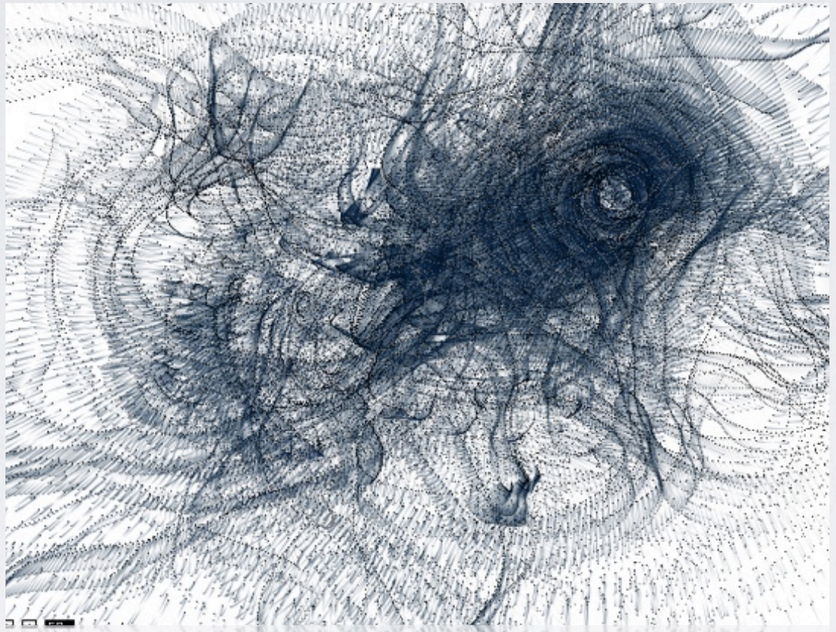
Utilizzeremo i primi due metodi

ESEMPI REALIZZAZIONI

http://processing.org/exhibition/

Shadow Monster

http://worthersoriginal.com/viki/#page=shadowmonsters



Withouttitle

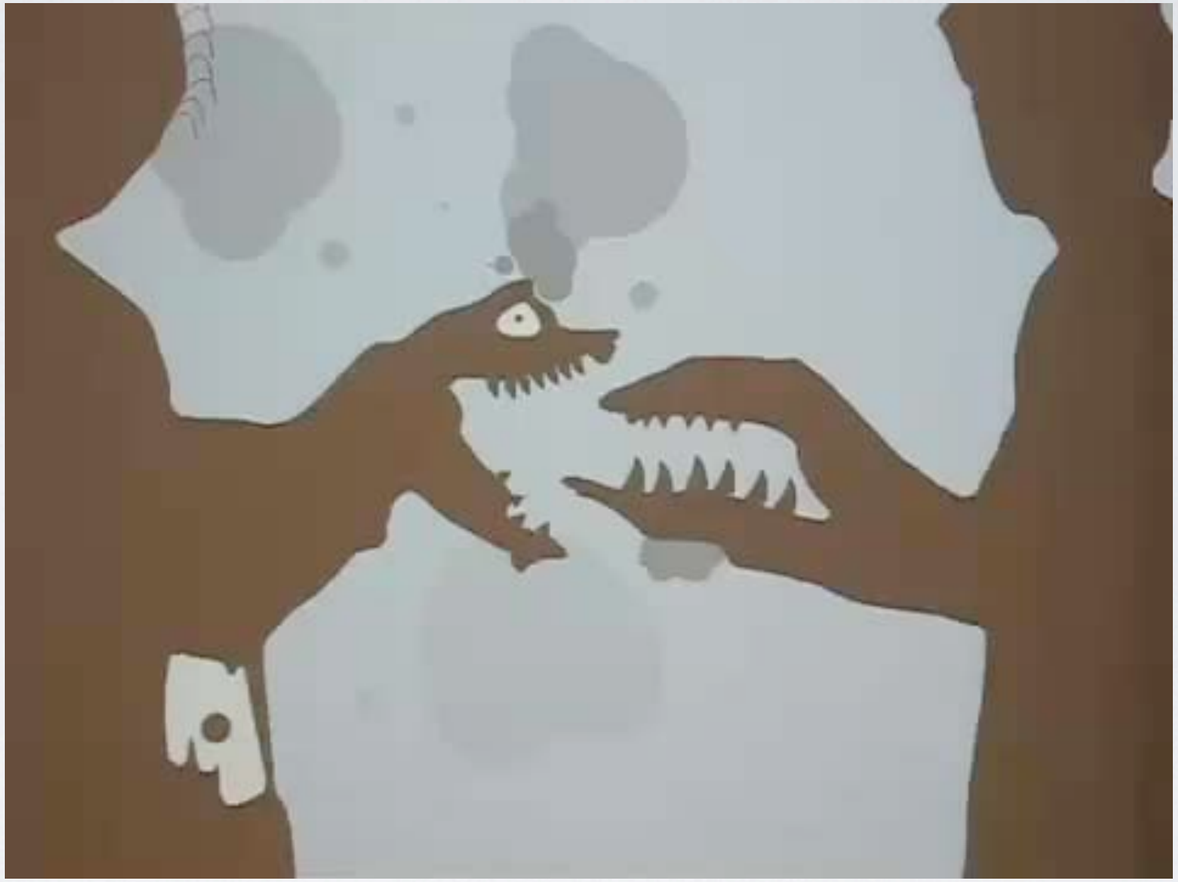
http://processing.org/exhibition/works/withouttitle/index_link.html

Metropop Denim

http://processing.org/exhibition/works/metropop/applet.html

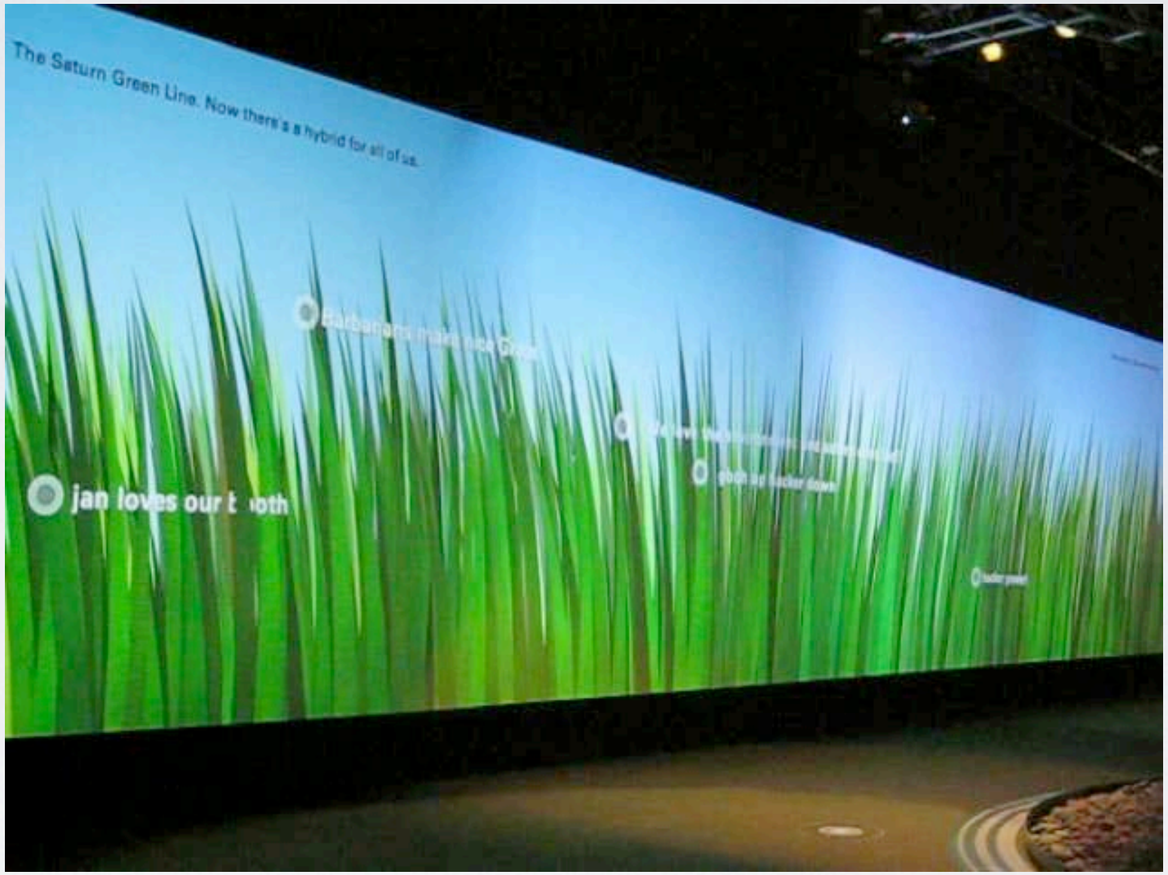
Shadow Monster

http://worthersoriginal.com/viki/#page=shadowmonsters



Grass

http://portfolio.barbariangroup.com/nextfest/index.html



Birds! Crow

http://www.flight404.com/blog/?p=99

PROCESSING

la programmazione

Una parte *dichiarativa*

La funzione **setup()**

La funzione **draw()**

```
// Parte dichiarativa

int radius = 30;

// Setup applicazione

void setup()
{

  size(400,400);

  smooth();

  frameRate(12);

}


//Disegno

void draw()

{

  ellipse(random(width),random(width),radius,radius);
```

variabili

in questa parte si dichiarano le variabili che verranno poi richiamate all'interno del programma

setup

all'interno del *setup* definiamo le proprietà dell'applicazione come ad esempio:
la dimensione della finestra
il framerate (in caso di animazioni)
il miglioramento dell'immagine
...

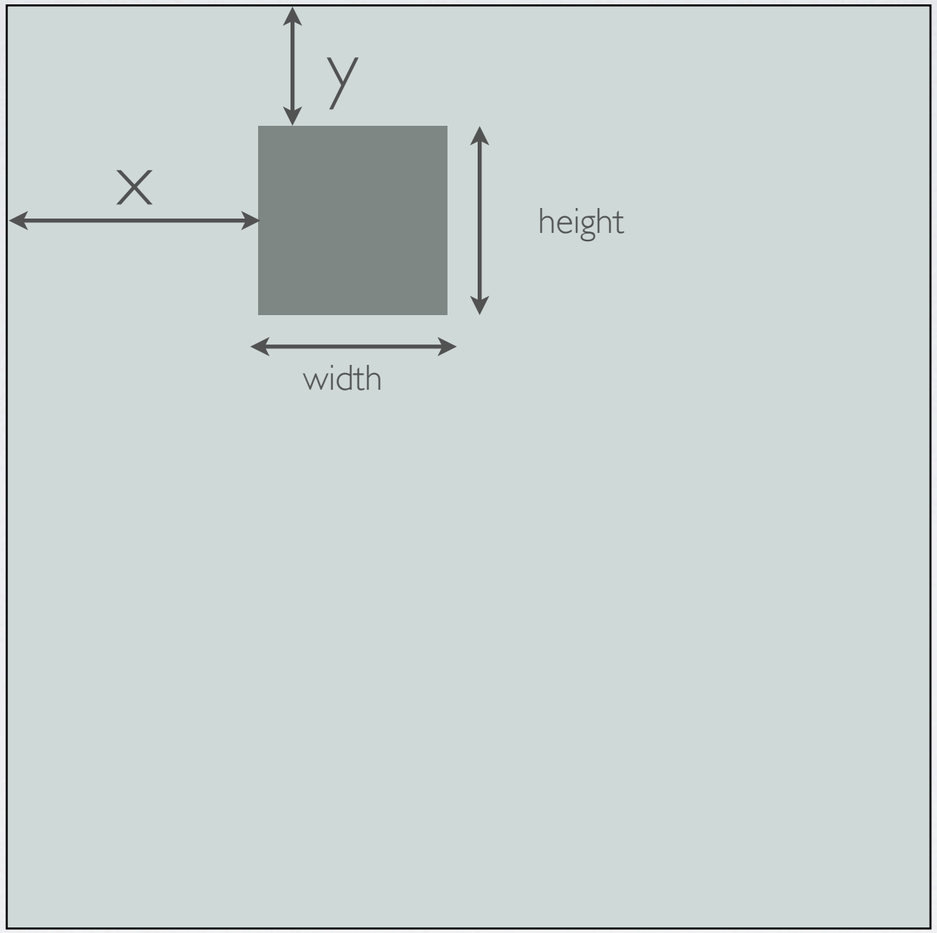
draw

all'interno di *draw* inseriamo ciò che vogliamo disegnare.
da ricordare, è che il disegno viene effettuato una volta per frame, nel caso del codice a lato il cerchio verranno disegnati 12 cerchi al secondo.

DISEGNAMO

RECT

rect(30, 20, 55, 55);
rect(x, y, width, height)



VOID SETUP

```
void setup(){
size(400, 400);
background(255,255,255);
}
```

{} contengono la funzione

()contengono le specifiche

; chiudono il comando

VOID DRAW

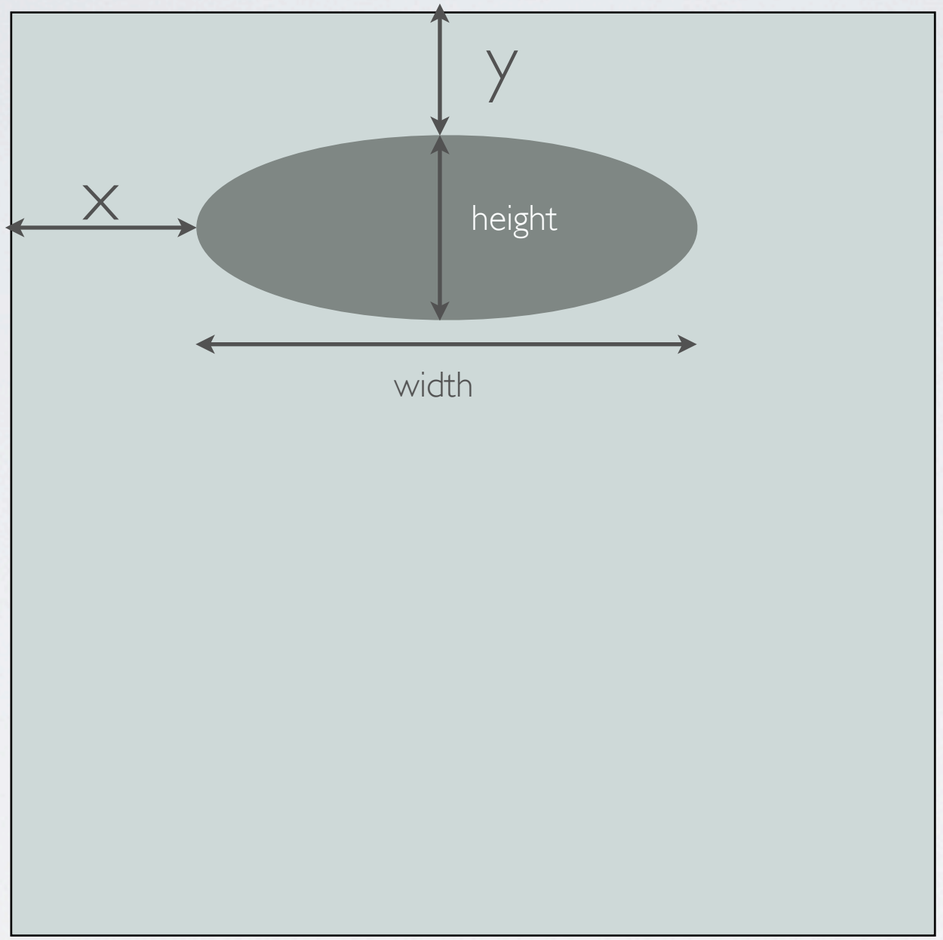
```
void draw(){

fill(255, 0, 0);

stroke(204, 104, 0);

rect(30, 20, 55, 55);

}
```

fill colore interno del rettangolo

stroke contorno del rettangolo

ELLIPSE

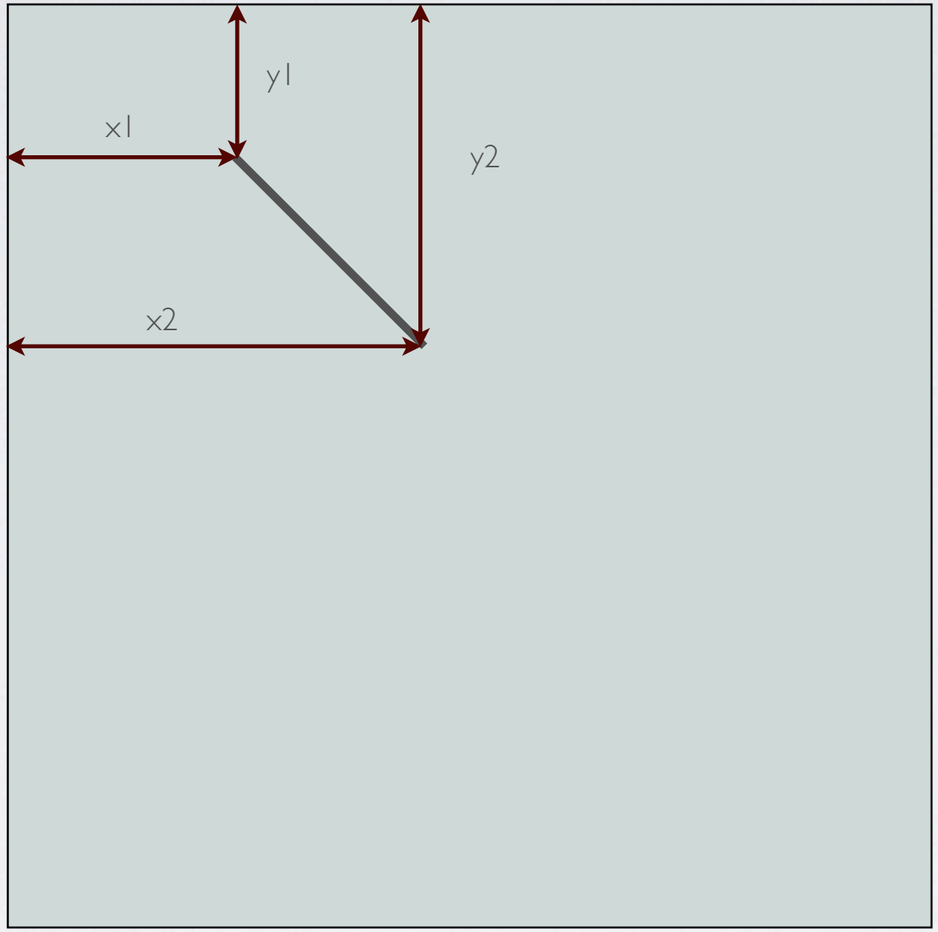
ellipse(56, 46, 55, 55);
ellipse(x, y, width, height)



```
void setup(){
size(400, 400);
background(255,0,0);
}
void draw(){
ellipse(30, 40, 56, 80);
}
```

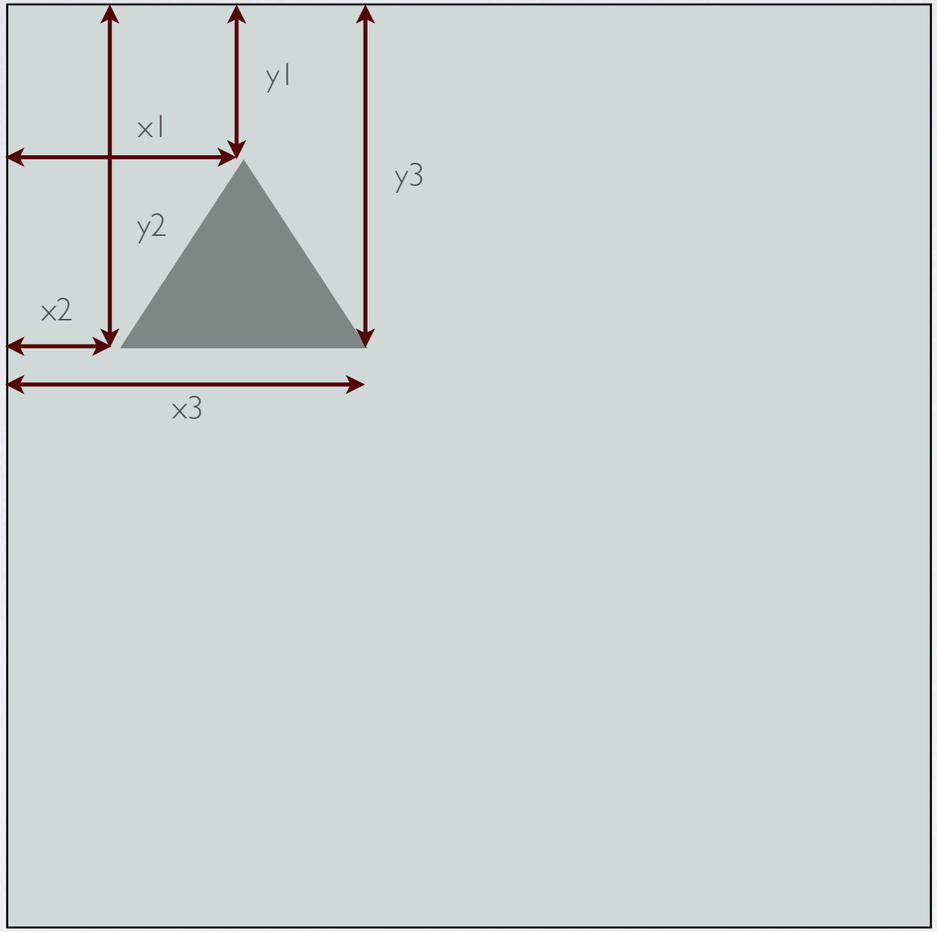
LINE

line(30, 20, 85, 75);
line(x1, y1, x2, y2)



```
void setup(){
size(400, 400);
background(255,0,0);
}
void draw(){
line(30, 20, 85, 75);
}
```

triangle(30, 75, 58, 20, 86, 75);
triangle(x1, y1, x2, y2, x3, y3);



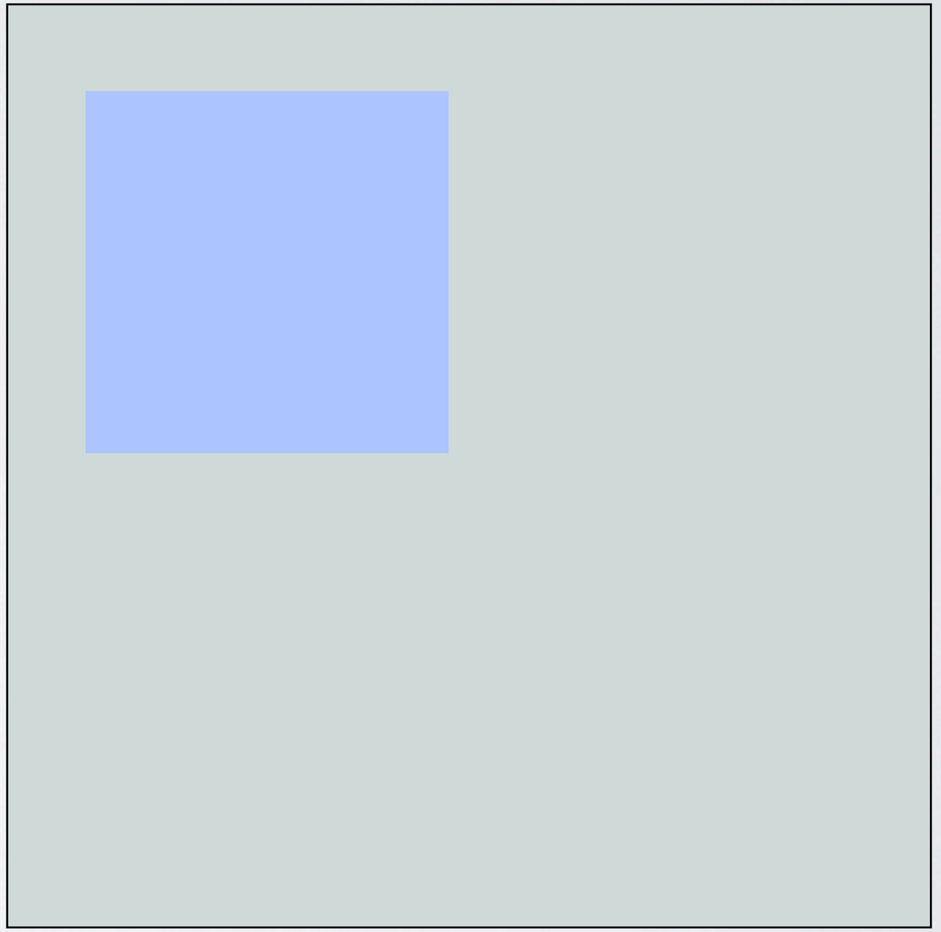
```
void setup(){
size(400, 400);
background(255,0,0);
}
void draw(){
triangle(30, 75, 58, 20, 86, 75);
}
```

QUADRATO COLORATO

COLORE INTERNO

fill(gray)
fill(gray, alpha)
fill(value1, value2, value3)
fill(value1, value2, value3, alpha)

```
fill(0,0,255);

noFill();
```



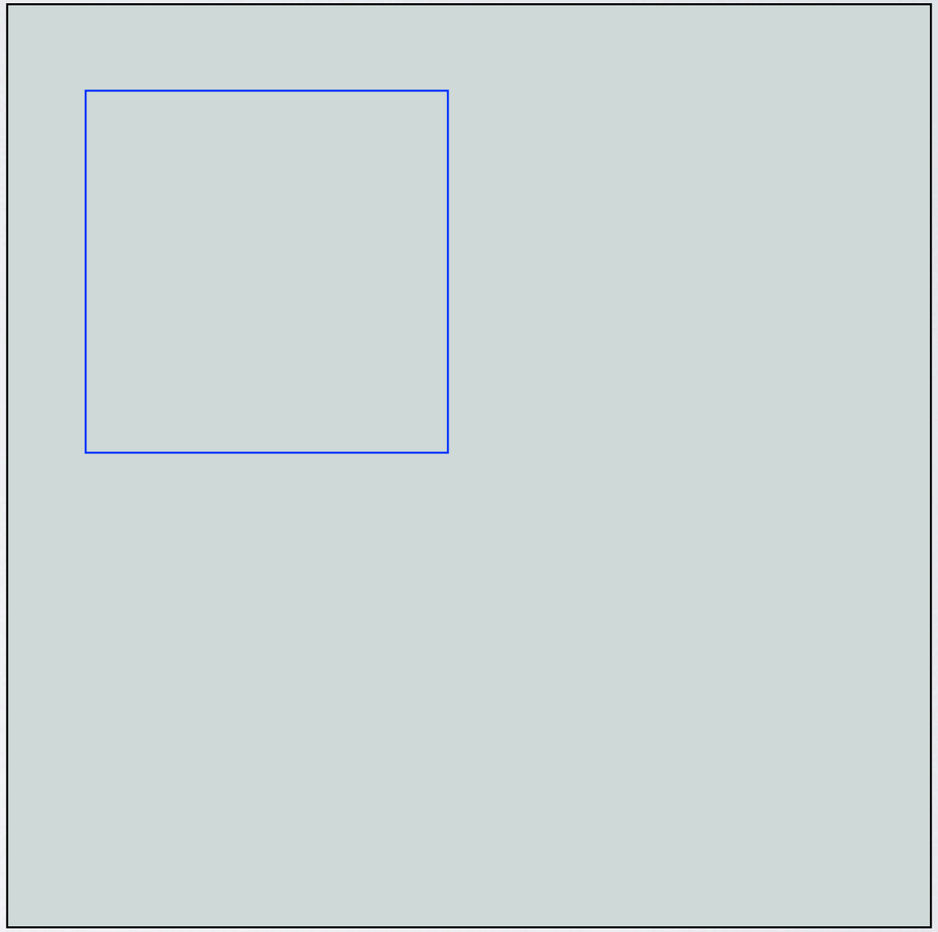
COLORE DEL FILETTO

stroke(gray)
stroke(gray, alpha)
stroke(value1, value2, value3)
stroke(value1, value2, value3, alpha)

stroke(0,0,255);

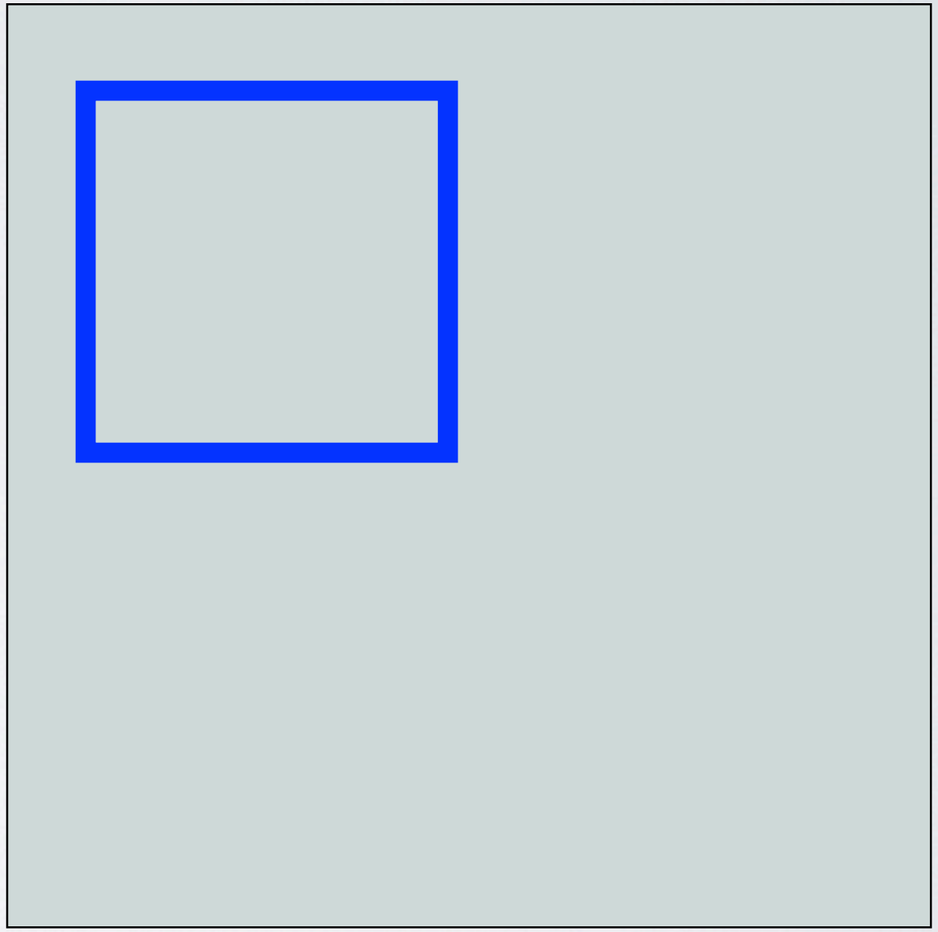
noStroke();

di default processing disegna il filetto ad 1 pixel di grandezza.



SPESSORE DEL FILETTO

strokeWeight(width)

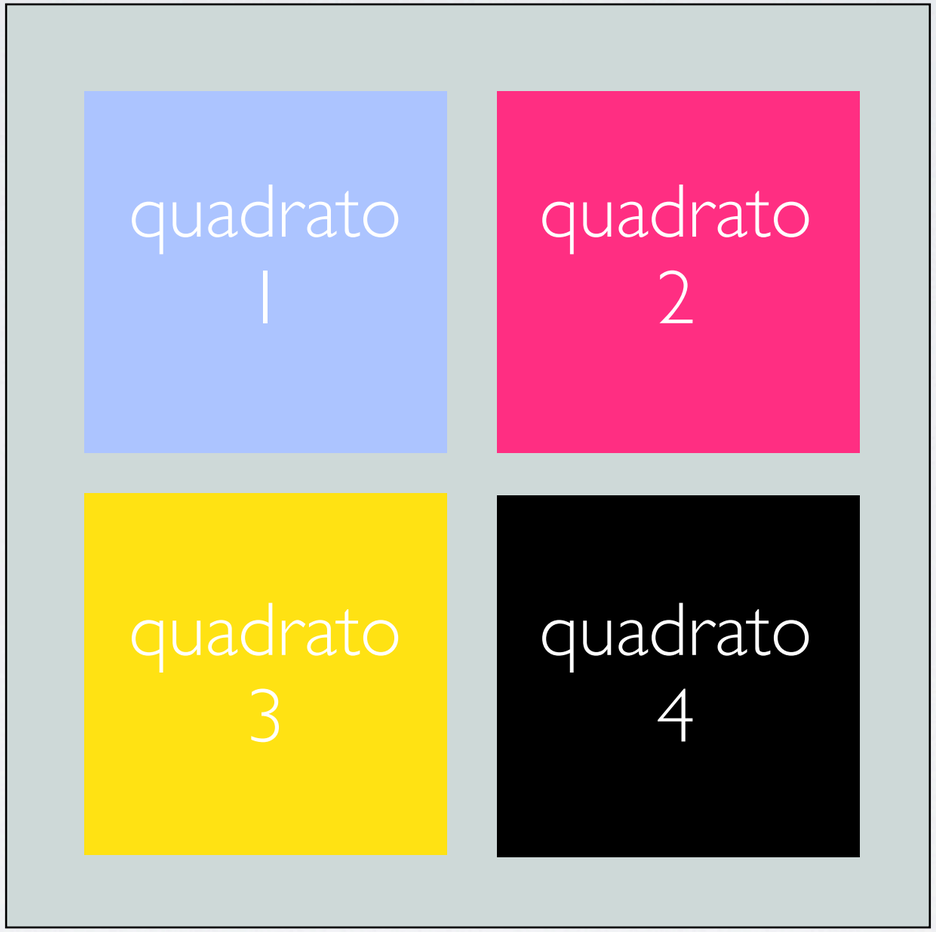


INTERATTIVITÀ

mouse hover

ESERCIZIO

```
void setup (){
size(600, 600);
background(255);

}
```




```
void draw (){
*/quadrato 1/*
???
*/quadrato 2/*
???
*/quadrato 3/*
???
*/quadrato 4/*

}
```

SE (*la X del mouse è all'interno del quadrato 1 && la Y del mouse è all'interno del quadrato 1*){

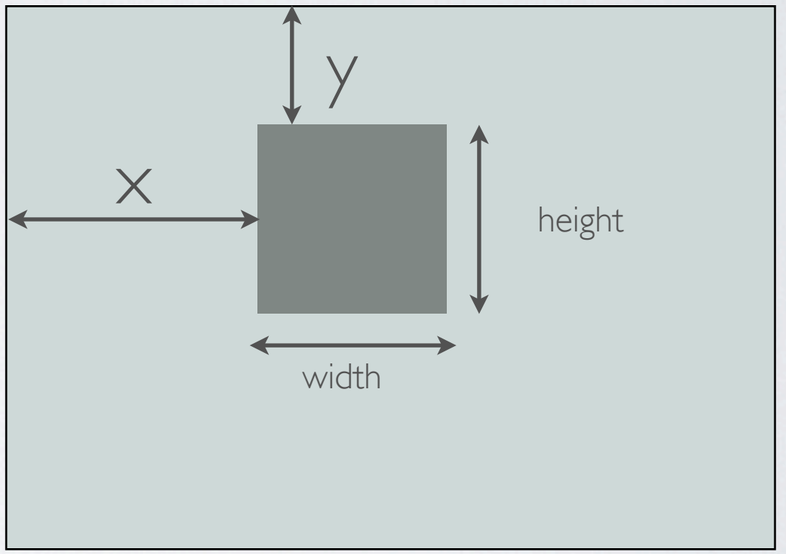
il quadrato 1 cambia colore}

ALTRIMENTI {

disegna il quadrato di colore ciano}

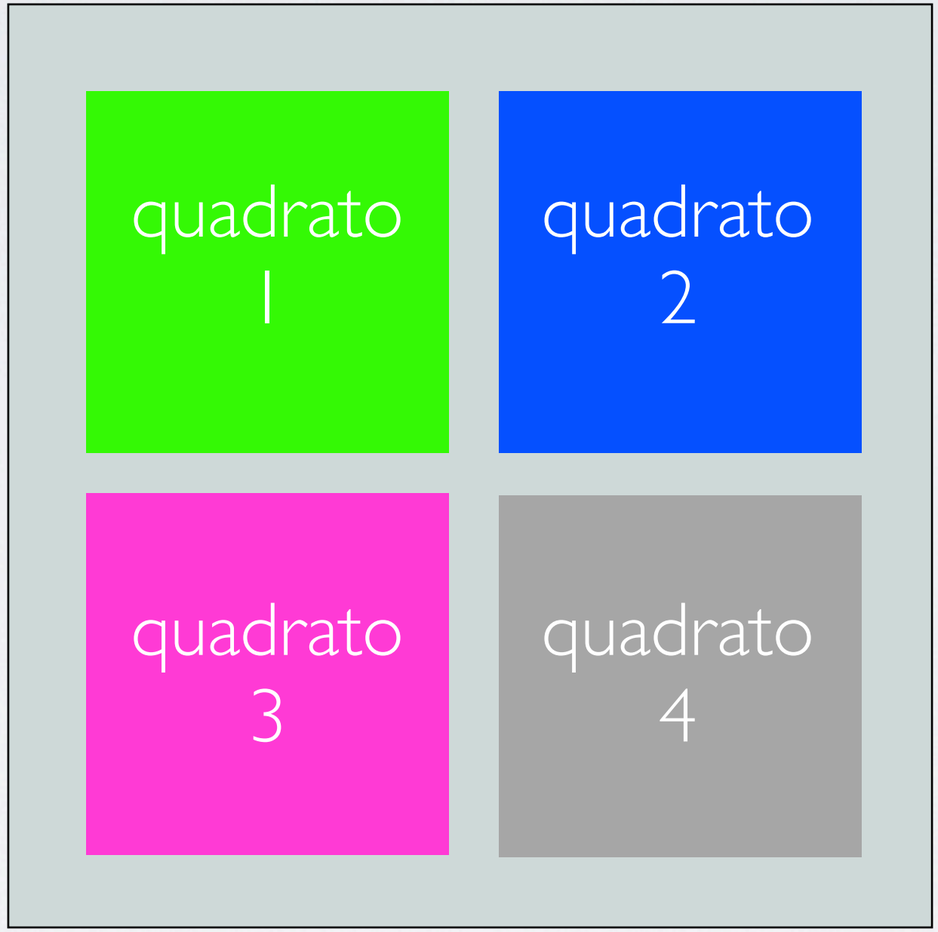
SE (*la X del mouse è all'interno del quadrato 1 && la Y del mouse è all'interno del quadrato 1*){
il quadrato 1 cambia colore}
ALTRIMENTI {
disegna il quadrato di colore ciano}

```
if(mouseX<=170 && mouseX>=20 && mouseY>=20 && mouseY<=170){
 fill(255,100,100);
  rect(20, 20, 150, 150);}else{
    fill(255,0,0);
  rect(20, 20, 150, 150);}
```



ESERCIZIO:

I QUADRATI CAMBIANO COLORE

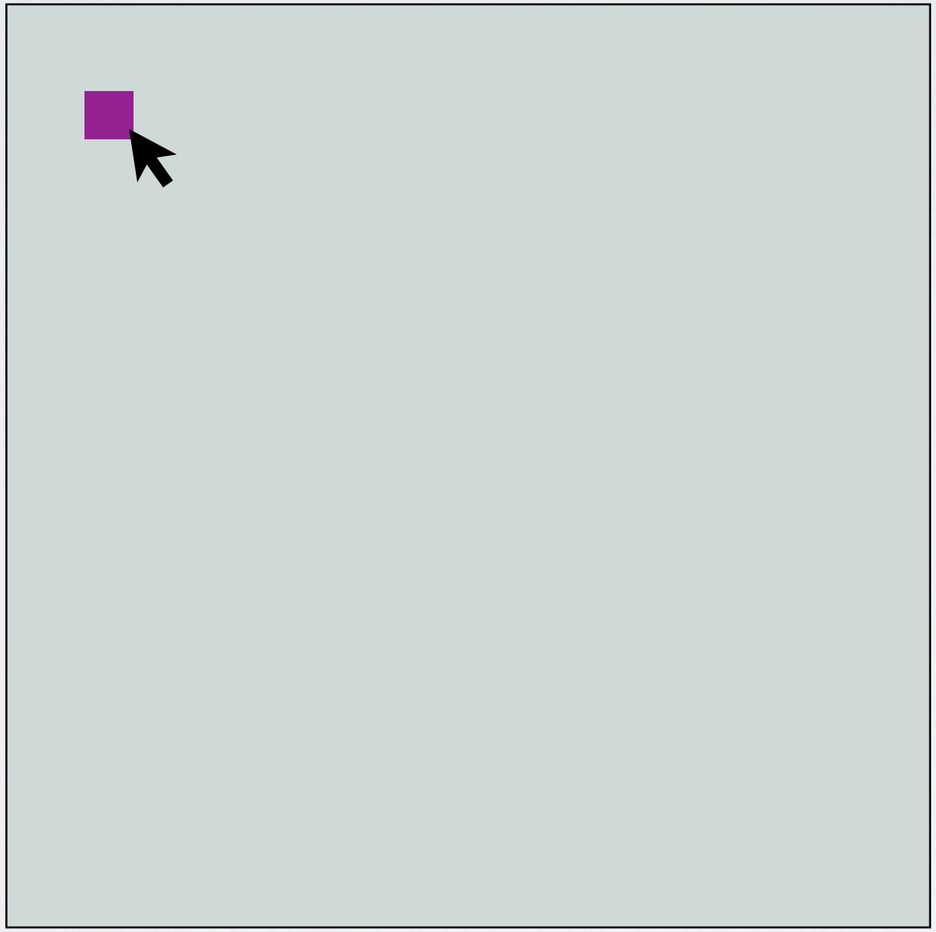


INTERATTIVITÀ

move mouse

ESERCIZIO

```
void setup (){
size(600, 600);
}
```

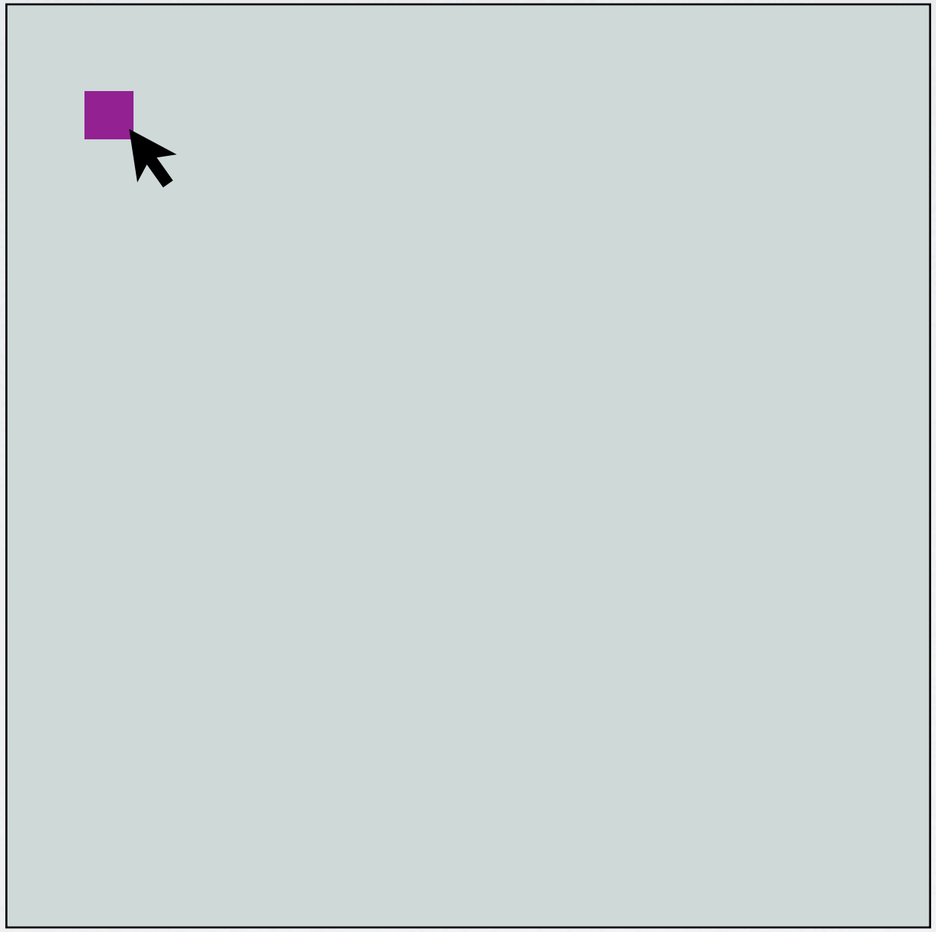


il quadrato segue il movimento del mouse

COME?

ESERCIZIO

```
void setup (){
size(600, 600);
}
```



il quadrato segue il movimento del mouse

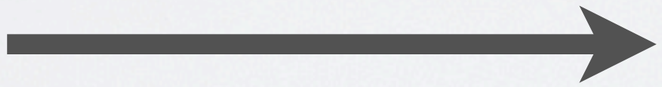
```
void draw(){
background(255,255,9);
rect(mouseX, mouseY, 30, 30);
}
```

VARIABILI

VARABILI

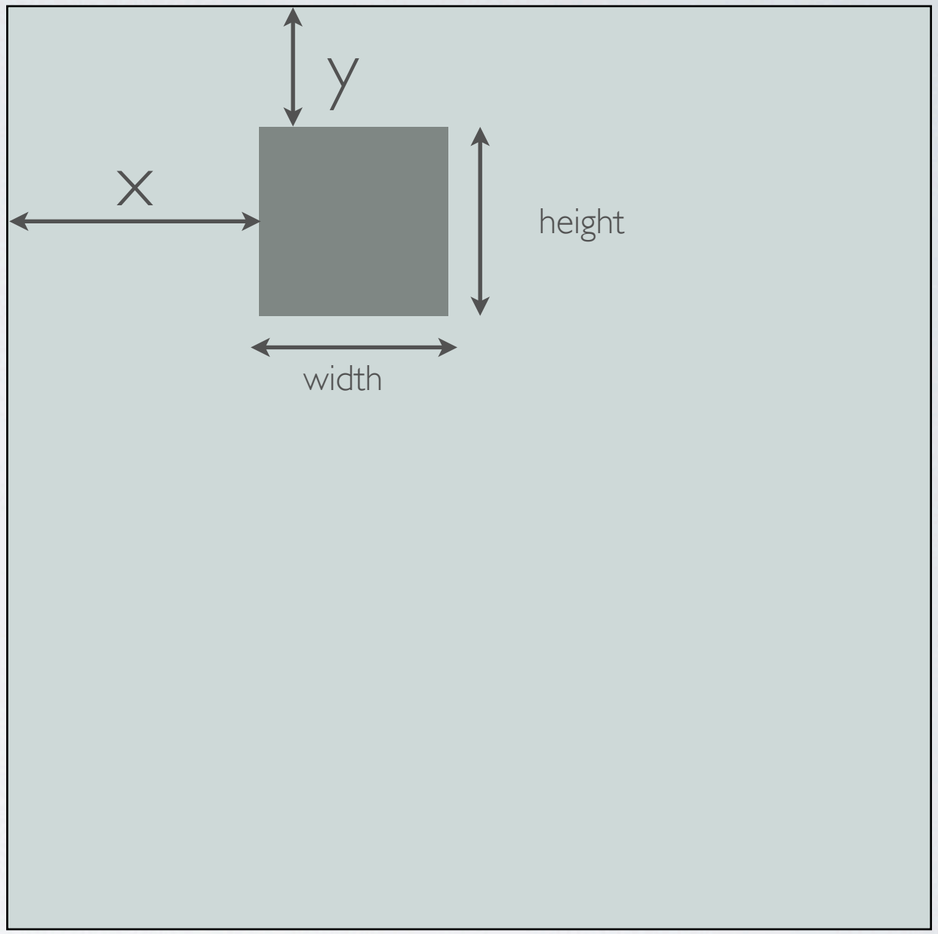
In informatica, una variabile identifica una porzione di memoria destinata a contenere dei dati, che possono essere modificati nel corso dell'esecuzione di un programma. Una variabile è spesso, ma non sempre, caratterizzata da un nome (inteso solitamente come una sequenza di caratteri e cifre).

int var
int var = value



int NABA;
int NABA = 10;

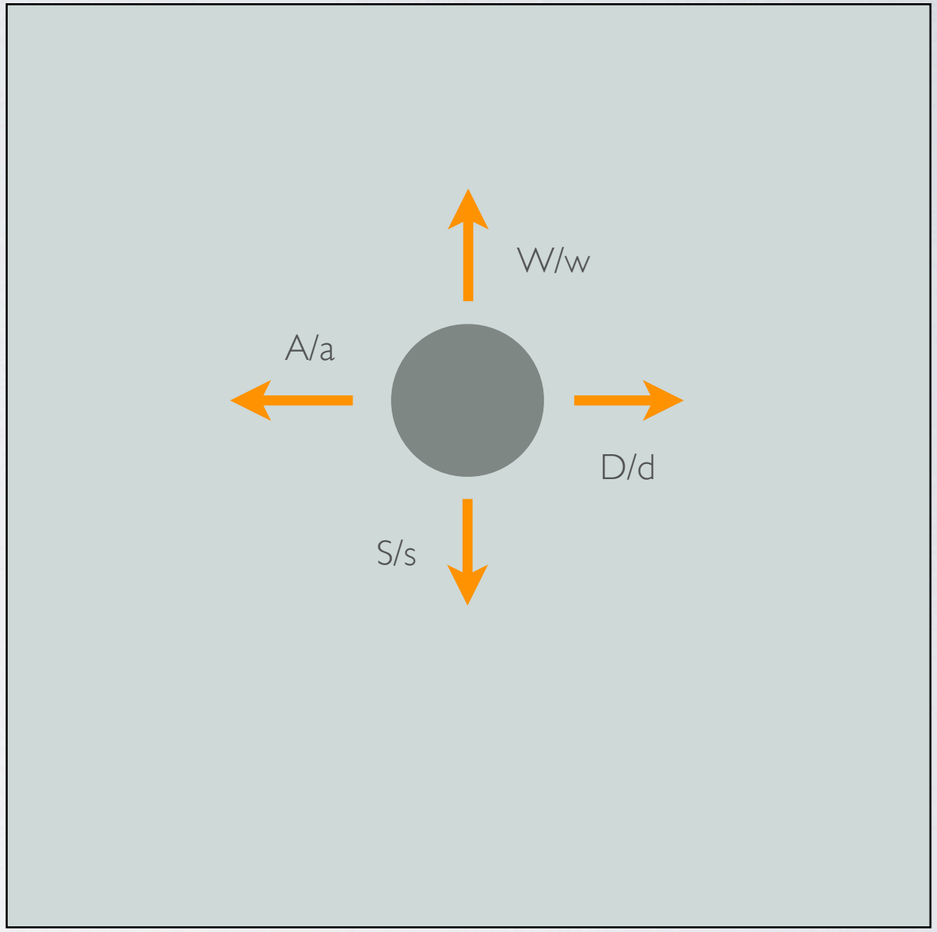
```
int x=30;
int y=20;

void setup(){
size(400, 400);
background(255,255,255);
}
void draw(){
fill(255, 0, 0);
stroke(204, 104, 0);
rect(x, y, 55, 55);
}
```



INTERATTIVITÀ

key pressed + int


esc
tab
Q
W
E
R
T
Y
U
I
O
P
delete
caps lock
A
S
D
F
G
H
J
K
L
return
shift
Z
X
C
V
B
N
M
shift
fn
control
option
command
command
option


W/w
A/a
D/d
S/s

IMPOSTIAMO UNA POSIZIONE

```
int pos1 = 50 ;
int pos2 = 50;
```

IMPOSTIAMO LA FINESTRA

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

DISEGNAMO UN CERCHIO

```
void draw (){
background(255);
ellipse(pos1, pos2, 50, 50);
...
```

SPOSTO A SINISTRA

se (il pulsante è premuto){
se(il pulsante è uguale ad ‘a’ oppure ad ‘A’)
la posizione l diminuisce di l pixel;
}

```
if (keyPressed){
if (key=='a' || key == 'A')
posl--;
}
```

ESERCIZIO

Scrivere il codice affichè il cerchio disegnato si sposti secondo le quattro direzioni (up, down, left, right) con i seguenti tasti(key):

a/A = left
s/S = down
d/D = right
w/W = up

INTERATTIVITÀ

Random | nascita casuale di cerchi

IMPOSTIAMO LE VARIABILI

impostiamo il raggio e l'incremento

```
int raggio=40;
int increment=50;
```

APRIAMO LA FINESTRA

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

RANDOM

```
random(high);
random(low, high);


---------------------------------
void draw(){

fill(random(255), random(255),increment, random(255));
noStroke();
ellipse(random(width), random(height), random(raggio), random(raggio));
noStroke();
}
```