

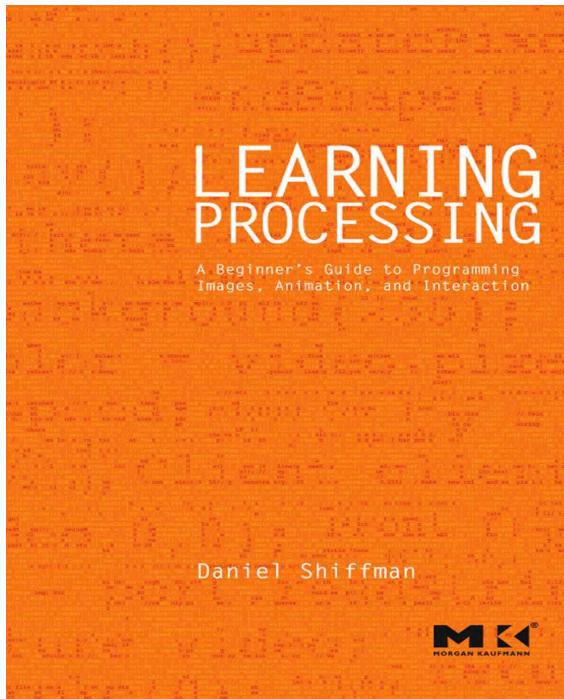


FONDAMENTI DI INFORMATICA

Alma Artis – Anno Accademico 2016/2017
Salvatore Rinzivillo (ISTI, CNR)

Introduzione a Processing

LIBRI E RIFERIMENTI



- Introduzione
- Capitolo 1-3
- Registrarsi al sito:
<http://almaartis.rinziv.it/>

Learning Processing– Second Edition
Daniel Shiffman
Available here: <http://learningprocessing.com/>



SOMMARIO

INFORMAZIONE

- Tutto ciò che può essere rappresentato all'interno di un computer
 - Numeri, caratteri, immagini, suoni
 - Comandi e sequenze di comandi che il calcolatore esegue per trasformare l'informazione
- Il **computer** (elaboratore elettronico) è lo strumento per rappresentare ed elaborare l'informazione

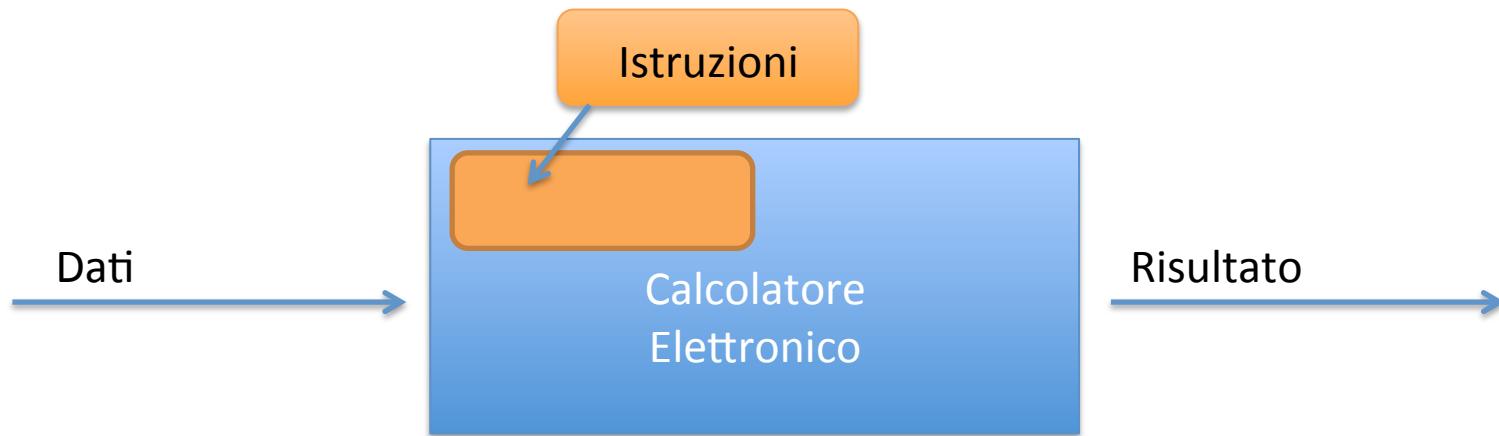
CALCOLATORE ELETTRONICO

Una macchina che:

- Ha un meccanismo di input per acquisire le richieste
- Ha un dispositivo per memorizzare le informazioni
- Ha la capacità di elaborare i dati
- Ha un dispositivo per comunicare al risposta

CALCOLATORE ELETTRONICO UNIVERSALE

- Il calcolatore può essere programmato per eseguire un insieme di azioni sui dati per risolvere un dato problema (o funzione)



FUNZIONI CALCOLABILI

- Di tutte le funzioni possibili, solo un sottoinsieme sono **calcolabili**, ovvero esiste un **algoritmo** che è in grado di fornire un risultato per i dati in input
- L'informatica si occupa di trovare risposte a domande del tipo:
 - Quante e quali sono le funzioni calcolabili?
 - Quale sono le funzioni che una data macchina può calcolare?
 - Esiste una macchina che calcoli tutte le (infinite) funzioni calcolabili?

ALGORITMO

- E' un insieme ordinato di azioni che risolve un dato problema (funzione) P
- L'esecuzione dell'algoritmo è affidata ad un **esecutore** (non necessariamente un calcolatore) in grado di decifrare le azioni nella sequenza
- Nella vita quotidiana tutti eseguiamo algoritmi:
 - Montare un mobile componibile
 - Cambiare la cartuccia della stampante
 - Prendere un caffè alla macchina a gettoni

ESERCIZIO

- Definire una attività quotidiana come un algoritmo astratto
 - Definire le istruzioni elementari
 - Definire l'algoritmo come sequenza di istruzioni elementari



LNGUAGGI DI PROGRAMMAZIONE

ALGORITMI E LINGUAGGI DI PROGRAMMAZIONE

- Nel caso di un elaboratore elettronico è necessario:
 - Conoscere l'insieme di istruzioni che è in grado di interpretare
 - Conoscere i tipi di dati che può rappresentare
- Questi due aspetti sono centrali nella definizione di un **Linguaggio di Programmazione**

ALGORITMI E PROGRAMMI

- Dato un **problema P**, la sua soluzione si identifica secondo la seguente procedura
 - Individuare un **metodo risolutivo**
 - Trasformare il metodo in un insieme ordinato di azioni: **algoritmo**
 - Rappresentare dati e azioni attraverso un formalismo comprensibile dal calcolatore (il linguaggio di programmazione): **programma**
- Una volta che il programma è stato creato, potrà essere utilizzato per risolvere ogni istanza del problema P



QUALE LINGUAGGIO DI PROGRAMMAZIONE?

- I linguaggi di programmazione sono tutti equivalenti
- Dati due linguaggi L_1 e L_2 e due programmi che risolvono la stessa funzione f nei due linguaggi $D_{L_1}(f)$ e $D_{L_2}(f)$
- Allora esiste una funzione calcolabile che traduce D_{L_1} in D_{L_2} e viceversa

IL LINGUAGGIO MACCHINA

- Il linguaggio direttamente eseguibile da un calcolatore si chiama linguaggio macchina
- E' un linguaggio poco comprensibile per gli umani
- Le operazioni disponibili sono molto semplici
- Sono specificate in notazione binaria: sequenze di 1 e 0
- Lo codifica di una funzione complessa richiede la scrittura di un lungo programma, a volte incomprensibile
- In caso di errori o malfunzionamenti, è difficile trovare gli eventuali errori

SOMMA DEI PRIMI 10 NUMERI

00110001 00000000 00000000

00110001 00000001 00000001

00110011 00000001 00000010

01010001 00001011 00000010

00100010 00000010 00001000

01000011 00000001 00000000

01000001 00000001 00000001

00010000 00000010 00000000

01100010 00000000 00000000

SOMMA DEI PRIMI 10 NUMERI

1. Store the number 0 in memory location 0.
2. Store the number 1 in memory location 1.
3. Store the value of memory location 1 in memory location 2.
4. Subtract the number 11 from the value in memory location 2.
5. If the value in memory location 2 is the number 0,
continue with instruction 9.
6. Add the value of memory location 1 to memory location 0.
7. Add the number 1 to the value of memory location 1.
8. Continue with instruction 3.
9. Output the value of memory location 0.

SOMMA DEI PRIMI 10 NUMERI

```
Set “total” to 0.  
Set “count” to 1.  
[loop]  
  Set “compare” to “count”.  
  Subtract 11 from “compare”.  
  If “compare” is zero, continue at [end].  
  Add “count” to “total”.  
  Add 1 to “count”.  
  Continue at [loop].  
[end]  
Output “total”.
```

Questo linguaggio primitivo permette di verificare se un valore è zero

Necessità di inserire etichette per controllare la sequenza di operazioni

SOMMA DEI PRIMI 10 NUMERI

```
var total = 0, count = 1;  
while (count <= 10) {  
    total += count;  
    count += 1;  
}  
console.log(total);  
// → 55
```

Non è necessario specificare
i salti all'esterno del ciclo di
somme

SOMMA DEI PRIMI 10 NUMERI

```
console.log(sum(range(1, 10)));  
// → 55
```

Introduzione di nuovi operatori: range e sum



PIXELS

COORDINATE

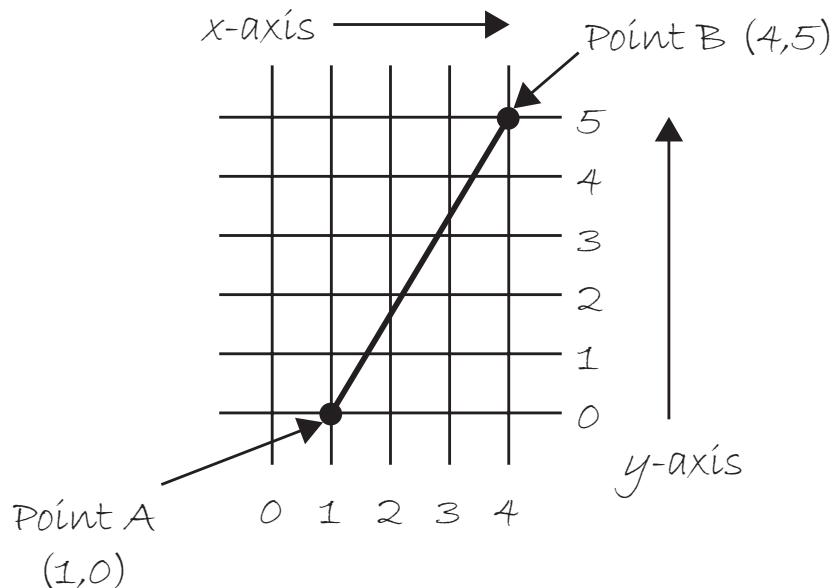


fig. 1.1

```
line(1,0,4,5);
```

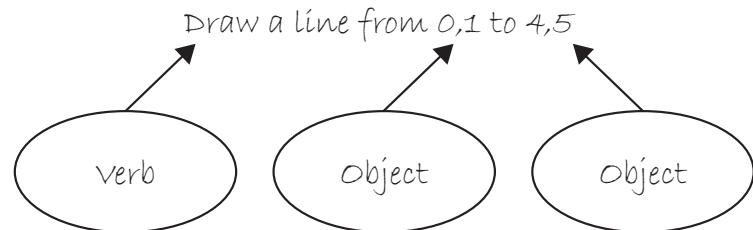
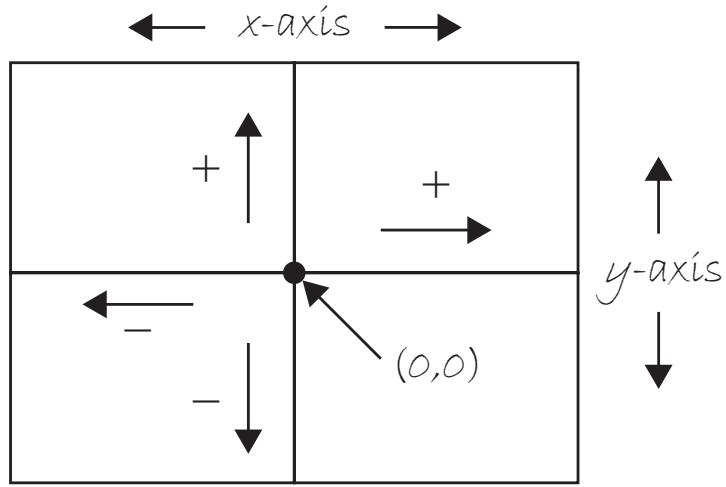


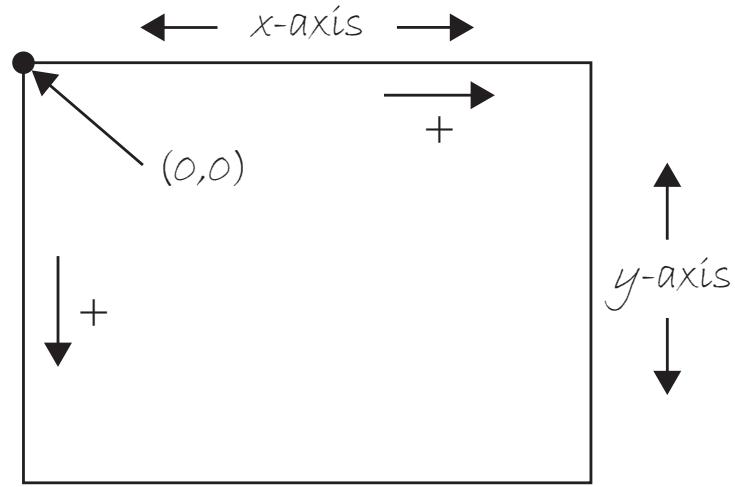
fig. 1.2

SISTEMA DI RIFERIMENTO



Eighth grade

fig. 1.3



Computer

FORME DI BASE



Point

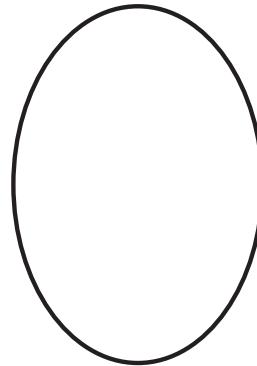
fig. 1.4



Line

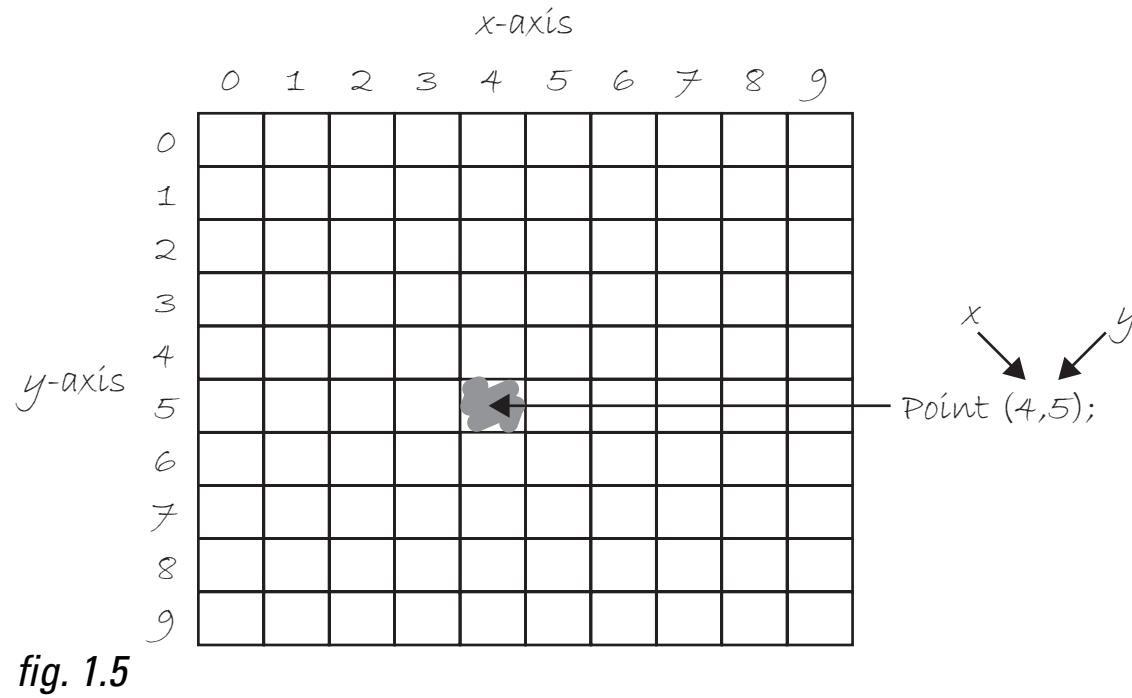


Rectangle



Ellipse

FORME DI BASE: POINT



FORME DI BASE: LINE

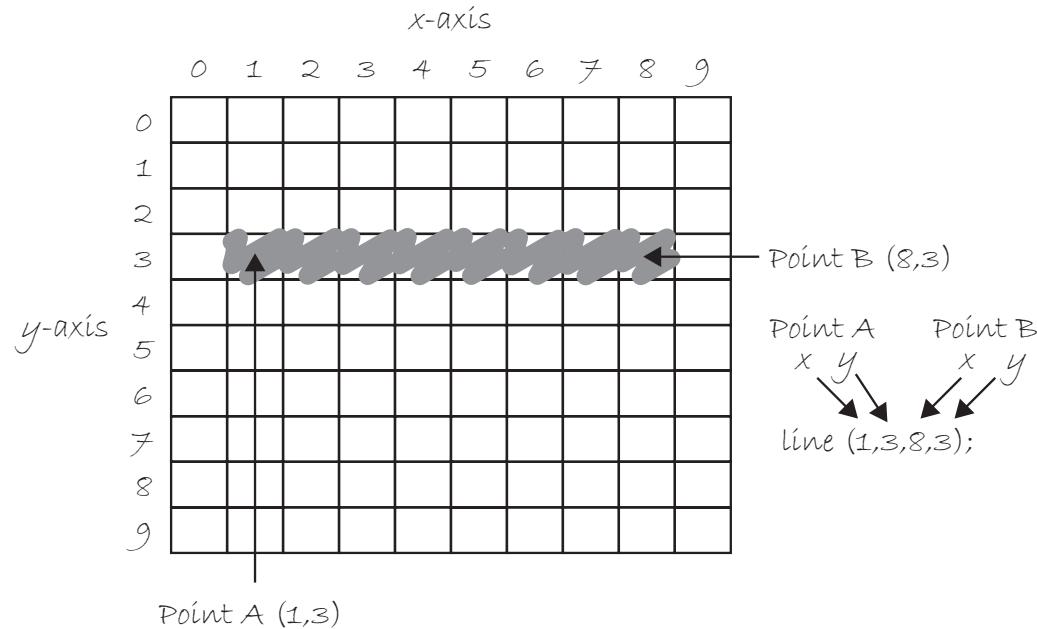


fig. 1.6

FORME DI BASE: RECT

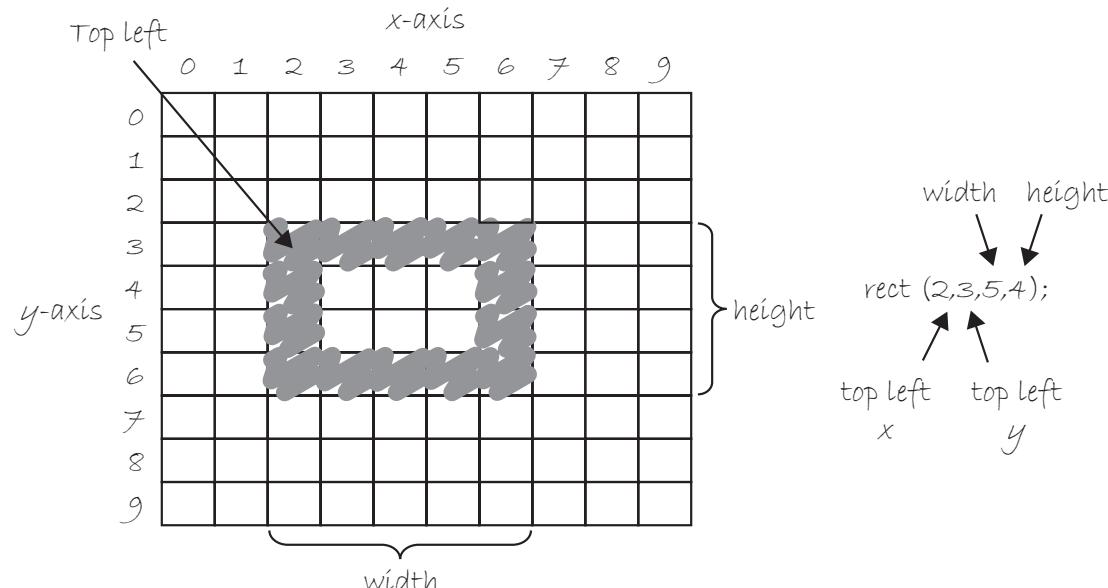


fig. 1.7

FORME DI BASE: RECT (2)

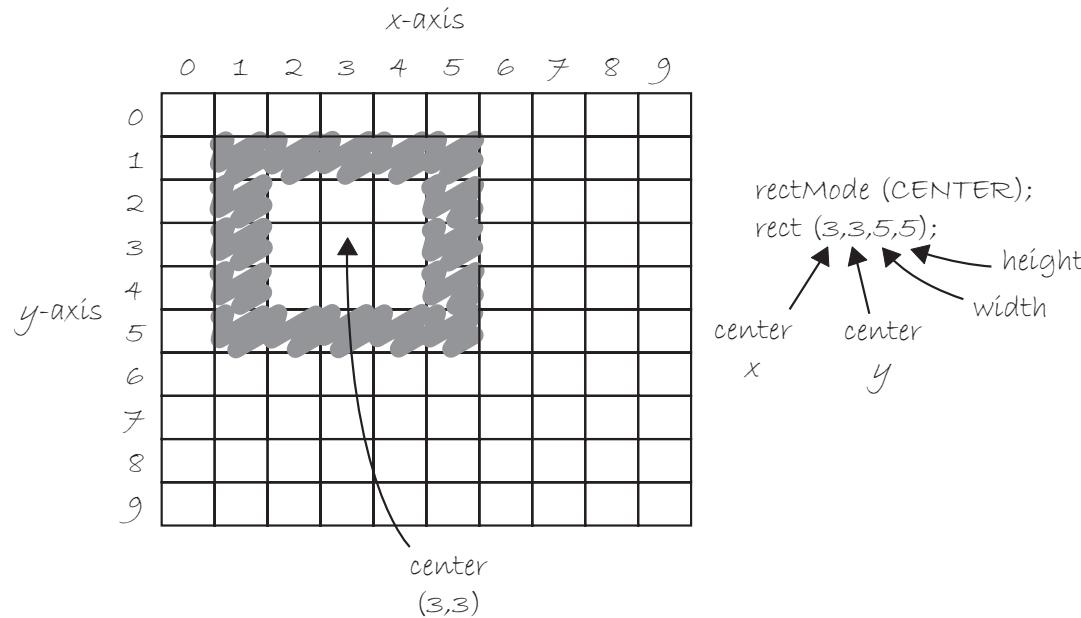


fig. 1.8

FORME DI BASE: RECT (3)

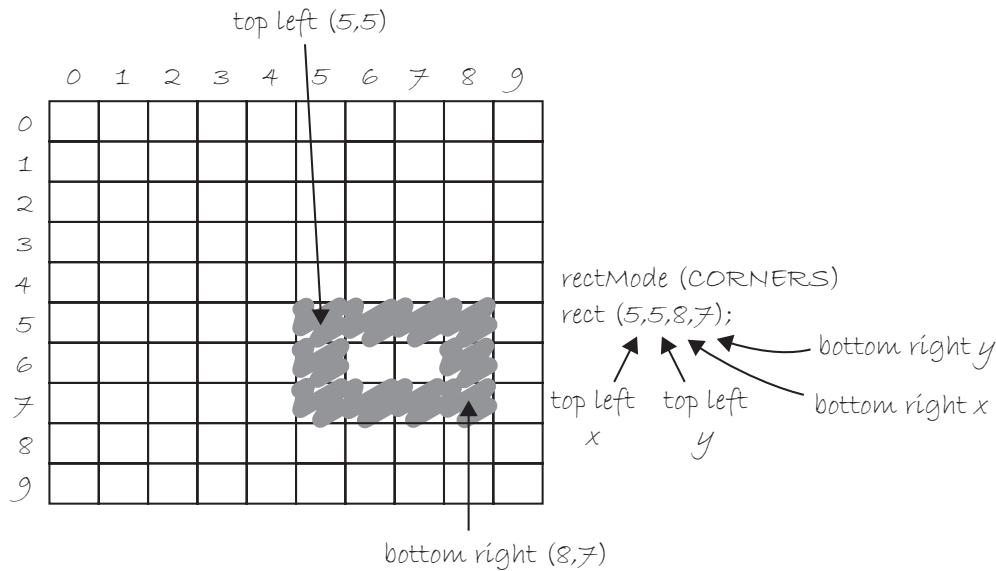
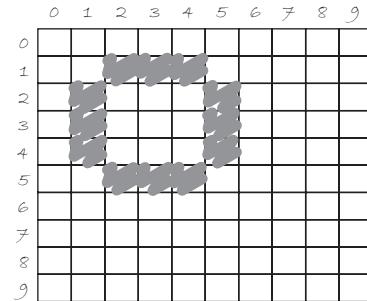
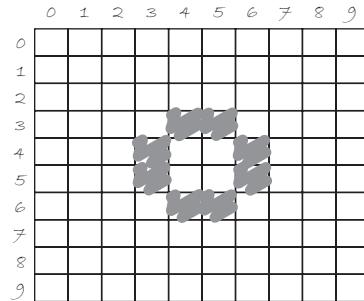


fig. 1.9

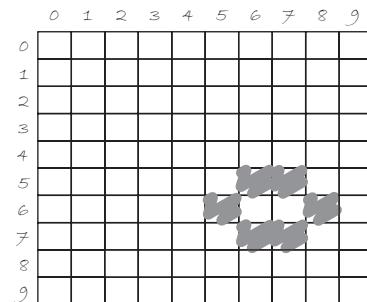
FORME DI BASE: ELLIPSE



```
ellipseMode (CENTER);  
ellipse (3,3,5,5);
```



```
ellipseMode (CORNER);  
ellipse (3,3,4,4);
```

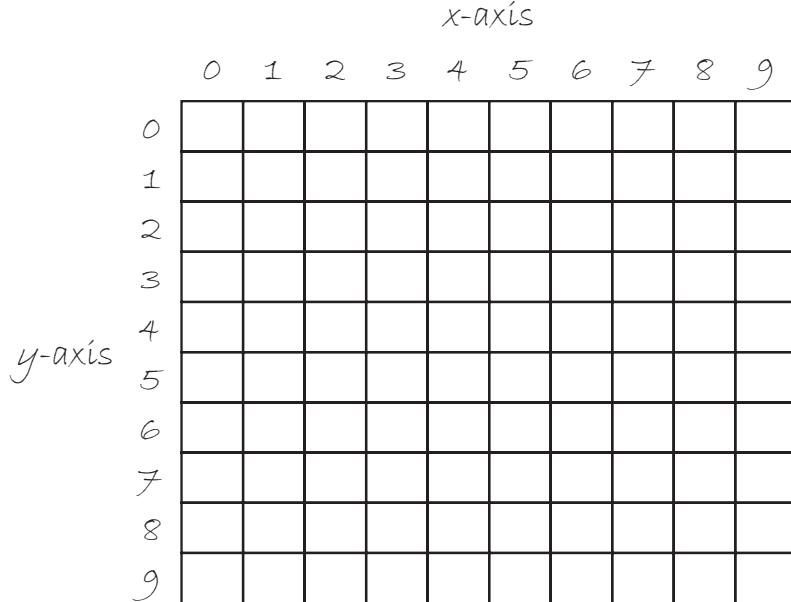


```
ellipseMode (CORNERS);  
ellipse (5,5,8,7);
```

fig. 1.10

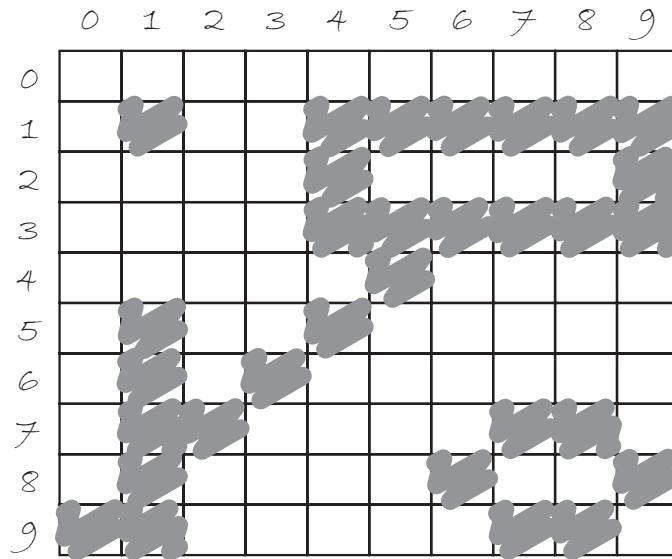
ESERCIZIO

```
line(0,0,9,6);  
point(0,2);  
point(0,4);  
rectMode(CORNER);  
rect(5,0,4,3);  
ellipseMode(CENTER);  
ellipse(3,7,4,4);
```



Disegnare le forme descritte dal codice a sinistra

ESERCIZIO



Note: There is more than one correct answer!

Scrivere un codice per riprodurre il diagramma



COLORI

SCALA DI GRIGIO: LUMINOSITÀ

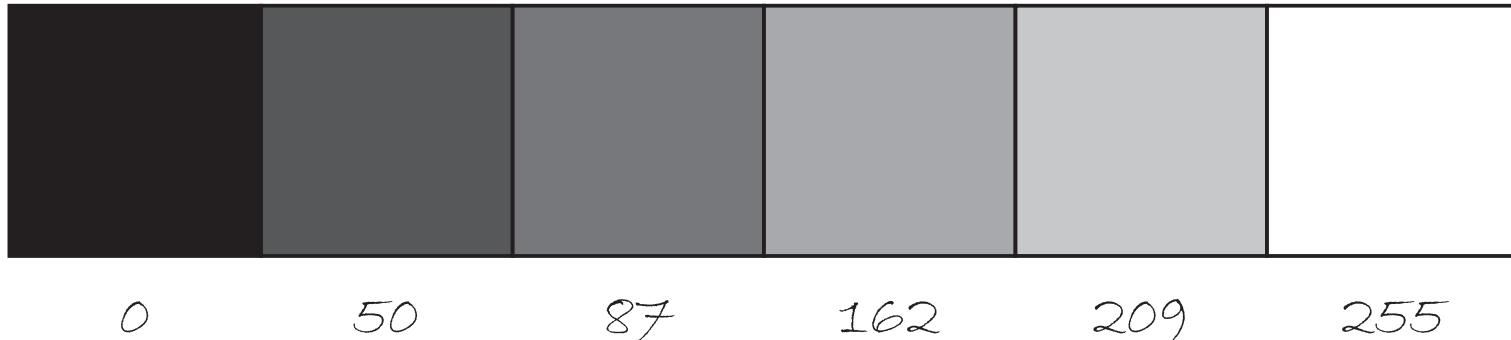


fig. 1.13

I livello di luminosità è rappresentato come un valore compreso tra 0 (nero) e 255 (bianco)

PARTI DI UNA FORMA GEOMETRICA

- Stroke: è la parte che definisce i contorni di una forma
 - Esempio: per un rettangolo sono i 4 lati
- Fill: è la parte interna della forma
 - Esempio: per un rettangolo, l'area racchiusa dai 4 lati

VALORI DI DEFAULT PER stroke E fill

```
rect(50,40,75,100);
```

The background color is gray.

The outline of the rectangle is black

The interior of the rectangle is white

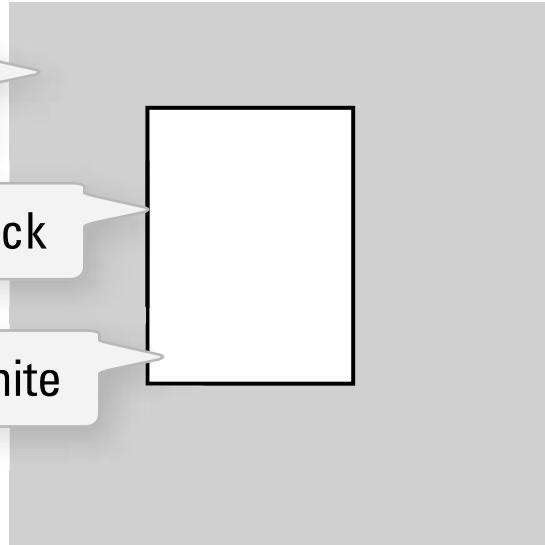


fig. 1.14

stroke **E** fill

```
background(255) ;  
stroke(0) ;  
fill(150) ;  
rect(50,50,75,100) ;
```

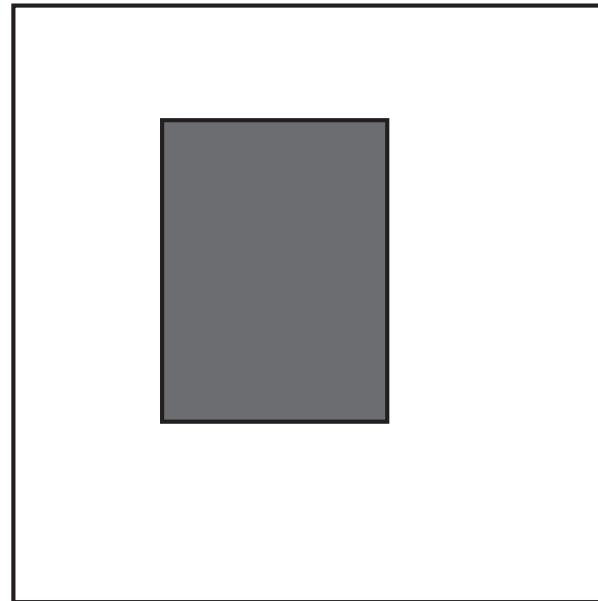


fig. 1.15

stroke & fill

```
background(255);  
stroke(0);  
noFill();  
ellipse(60,60,100,100);
```

nofill() leaves the shape
with only an outline

If we draw two shapes at one time, *Processing* will always use the most recently specified ***stroke()*** and ***fill()***, reading the code from top to bottom. See Figure 1.17.

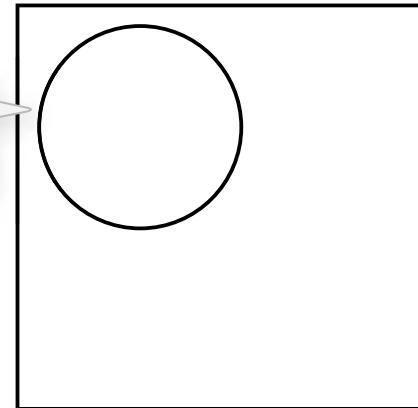


fig. 1.16

stroke & fill

```
background(150);  
stroke(0);  
line(0, 0, 100, 100);  
stroke(255);  
noFill();  
rect(25, 25, 50, 50);
```

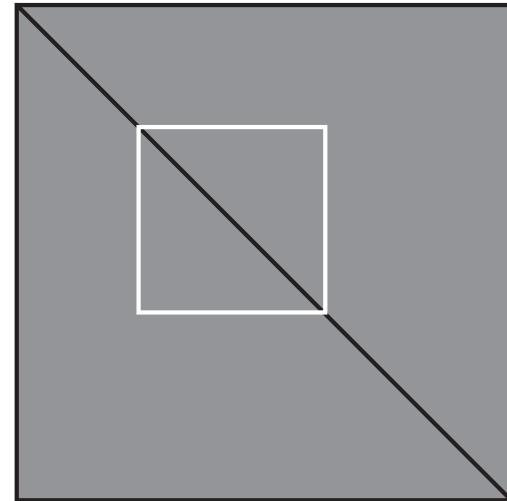
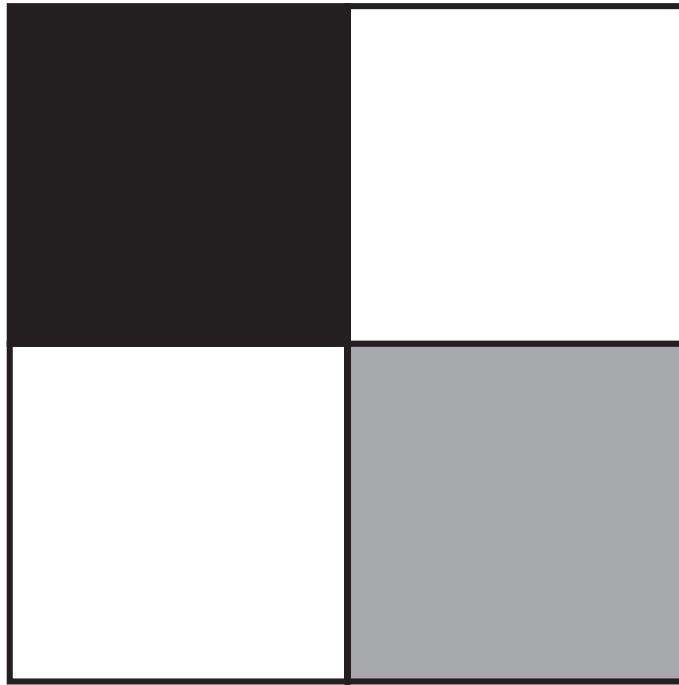


fig. 1.17

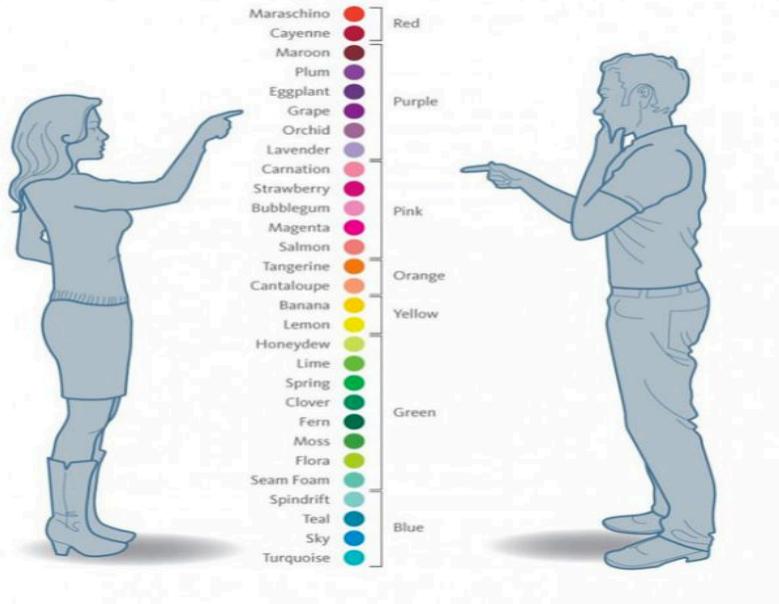
ESERCIZIO



- Scrivere le istruzioni per creare il diagramma a sinistra

HOW MANY COLOR?

Female



Male

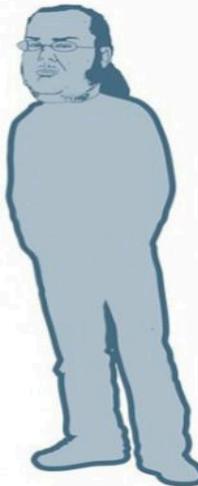


Dog



Programmer

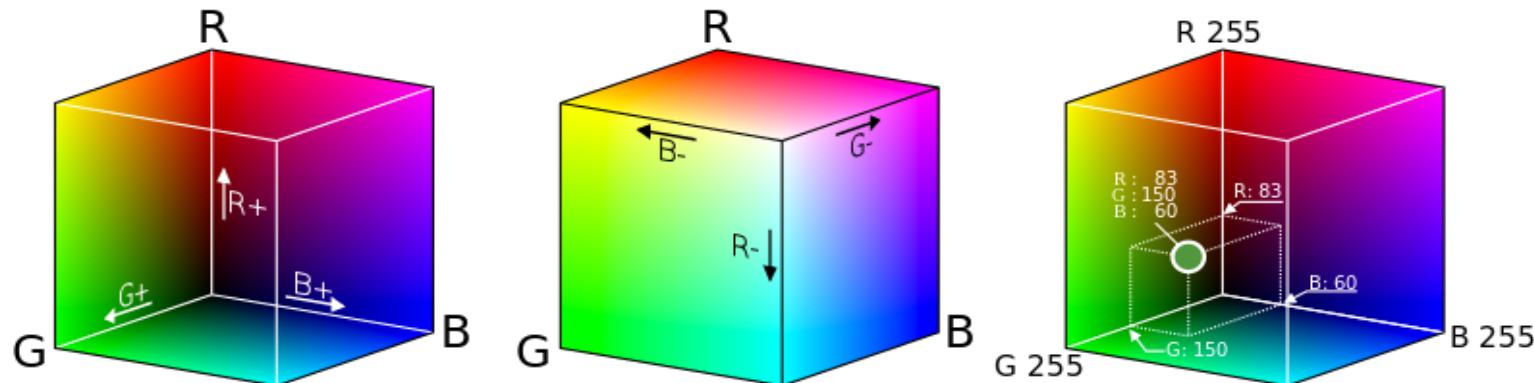
Gray ● #f94433
Gray ● #ac203b
Gray ● #85343d
Gray ● #874994
Gray ● #663c84
Gray ● #8c2590
Gray ● #a16799
Gray ● #af99c7
Gray ● #f38da3
Gray ● #d2157b
Gray ● #ec90b7
Gray ● #e90086
Gray ● #f57d7e
Gray ● #f27727
Gray ● #fc9b7b
Gray ● #7d305
Gray ● #f1e311
Gray ● #ccdf62
Gray ● #68bd46
Gray ● #0aae4f
Gray ● #069665
Gray ● #057054
Gray ● #3ba246
Gray ● #abcf37
Gray ● #68c3b2
Gray ● #8bccd0
Gray ● #0687a7
Gray ● #078dca
Gray ● #0fb8b5



WeKnowMemes

RGB COLOR MODEL

- Basato sui colori primari
- È un modello additivo: il colore risultante è la somma dei componenti di base



RGB COLOR MODEL

- I valori R,G,B possono essere espressi come numeri nell'intervallo [0,1]
- Alcune applicazioni (e anche processing) usano l'intervallo [0,255]
- Non è un modello intuitivo. Come definire i valori per un marrone chiaro?

COLORI RGB

Example 1-3: RGB color

```
background(255) ;  
noStroke() ;  
  
fill(255, 0, 0) ;  
ellipse(20, 20, 16, 16) ;  
  
fill(127, 0, 0) ;  
ellipse(40, 20, 16, 16) ;  
  
fill(255, 200, 200) ;  
ellipse(60, 20, 16, 16) ;
```

- Bright red
- Dark red
- Pink (pale red).



fig. 1.18

TRASPARENZA

Example 1-4: Alpha transparency

```
background(0);  
noStroke();
```

```
fill(0,0,255);  
rect(0,0,100,200);
```

No fourth argument means 100% opacity.

```
fill(255,0,0,255);  
rect(0,0,200,40);
```

255 means 100% opacity.

```
fill(255,0,0,191);  
rect(0,50,200,40);
```

75% opacity

```
fill(255,0,0,127);  
rect(0,100,200,40);
```

50% opacity

```
fill(255,0,0,63);  
rect(0,150,200,40);
```

25% opacity



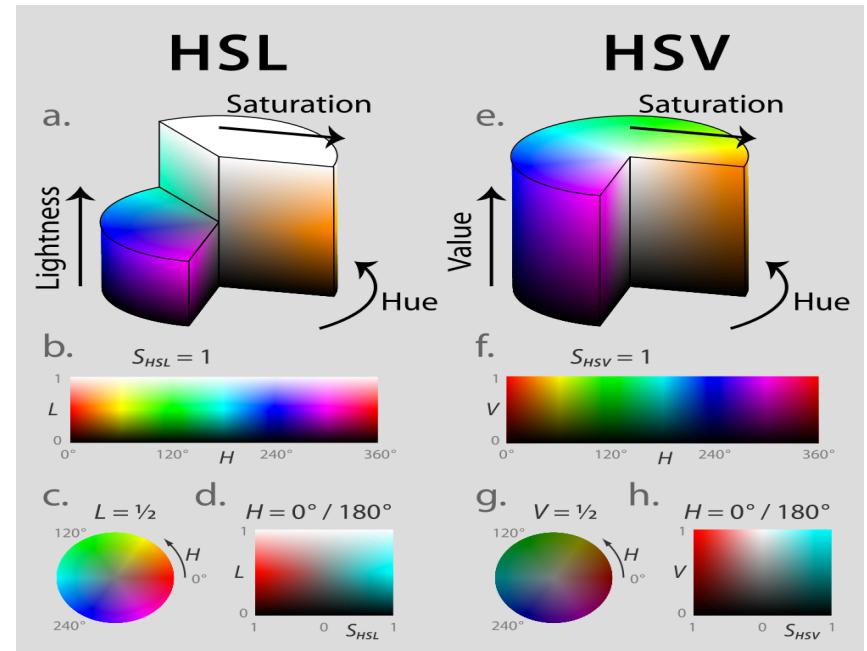
fig. 1.20

COLOR MODE

- I range di colori sono riconosciuti solo come intervalli tra 0 e 255
- E' possibile definire una mappa diversa utilizzando l'istruzione `colorMode`
 - `colorMode(RGB,100)`
- Alternative color model: HSB
 - Hue: il tipo di colore da usare (varia tra 0 e 360)
 - Saturation: vivicità del colore (varia tra 0 e 100)
 - Brightness: luminosità (varia tra 0 e 100)

HSV COLOR MODEL

- Basato sui concetti intuitivi di
 - tinta
 - saturazione
 - Intensità o valore



"Hsl-hsv models" by Jacob Rus - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Hsl-hsv_models.svg#/media/File:Hsl-hsv_models.svg

ESERCIZIO

Descrivere una creatura usando semplici forme geometriche

Example 1-5: Zoog

```
ellipseMode(CENTER);  
rectMode(CENTER);  
stroke(0);  
fill(150);  
rect(100,100,20,100);  
fill(255);  
ellipse(100,70,60,60);  
fill(0);  
ellipse(81,70,16,32);  
ellipse(119,70,16,32);  
stroke(0);  
line(90,150,80,160);  
line(110,150,120,160);
```

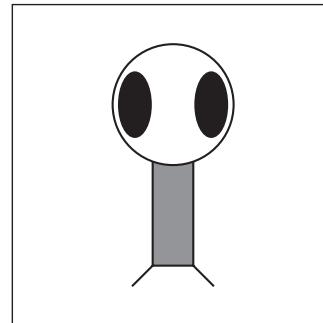


fig. 1.21



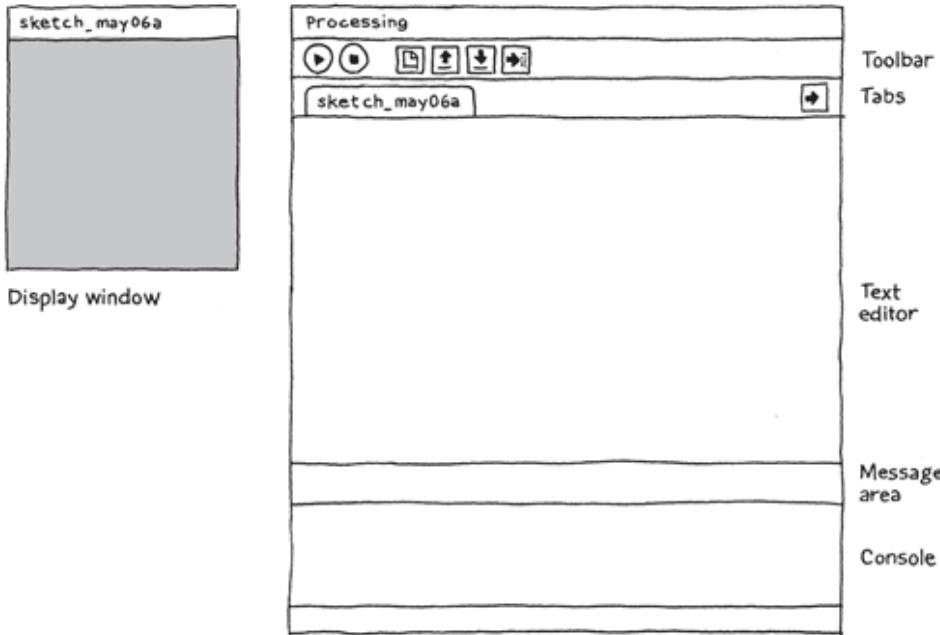
A decorative header pattern at the top of the page consists of six horizontal panels. From left to right: 1) A panel of light blue hexagons on a dark navy background. 2) A panel featuring nested diamond shapes in light blue, medium blue, and white. 3) A panel with a grid of triangles in white, medium blue, and dark navy. 4) A panel of light blue hexagons on a dark navy background. 5) A panel featuring nested diamond shapes in light blue, medium blue, and white. 6) A panel with a grid of triangles in white, medium blue, and dark navy.

INTRODUCING PROCESSING

PROCESSING.ORG INSTALLATION

- Download IDE from <http://processing.org>
- Expand archive file
- Execute the application

ANATOMY OF CODE EDITOR



ANATOMY OF CODE EDITOR



Execute current Sketch



Stop execution of current Sketch



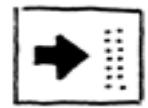
Save current Sketch



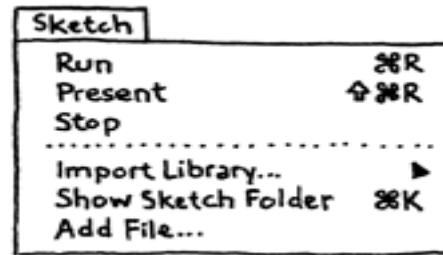
Load an existing Sketch from disk



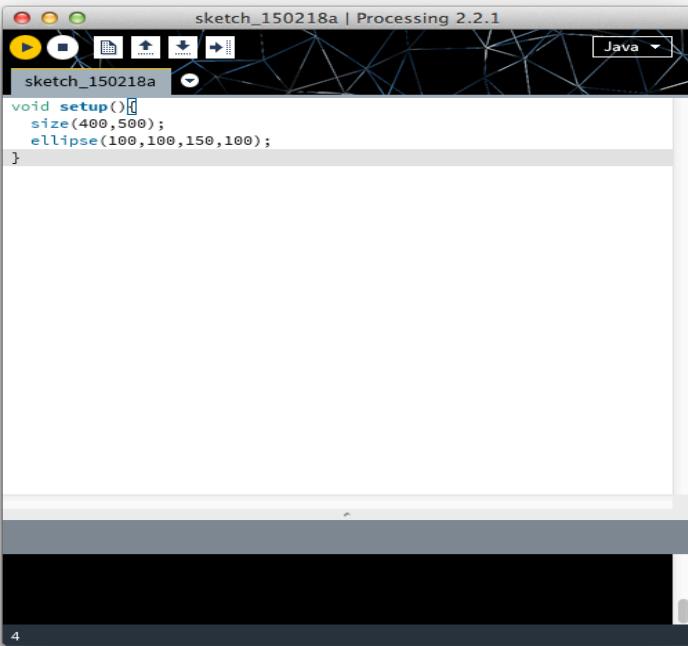
Start a new Sketch



Export current Sketch as a standalone app



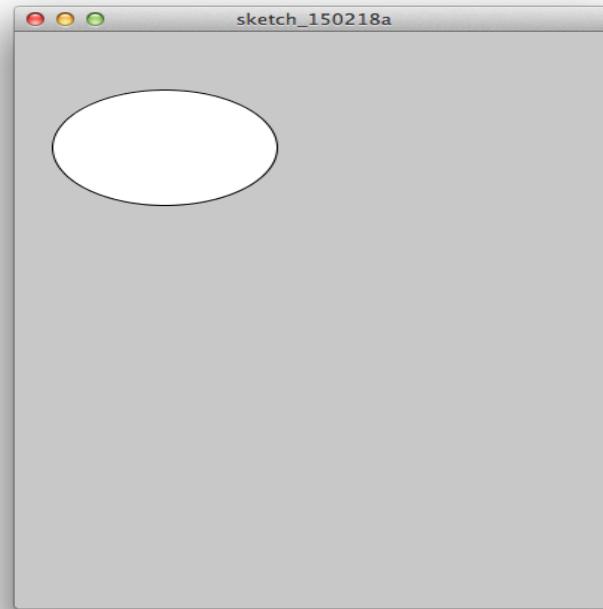
OUR FIRST SKETCH



The screenshot shows the Processing 2.2.1 IDE interface. The title bar reads "sketch_150218a | Processing 2.2.1". The toolbar contains various icons for file operations like Open, Save, and Run. A dropdown menu "Java" is open. The code editor window displays the following Java code:

```
void setup(){  
    size(400,500);  
    ellipse(100,100,150,100);  
}
```

The code defines a sketch with a setup function that creates a window of size 400x500 and draws an ellipse at coordinates (100, 100) with a width of 150 and a height of 100.



CODICE IN PROCESSING

- Possiamo scrivere tre tipi di istruzioni
 - Chiamare delle funzioni
 - Assegnare un valore ad una variabile
 - Strutture di controllo

The diagram shows a single line of Processing code: `Line (0,0,200,200);`. A bracket labeled "Arguments in parentheses" spans from the opening parenthesis to the closing parenthesis. An arrow labeled "Function name" points to the word "Line". Another arrow labeled "Ends with semi-colon" points to the final semi-colon.

Line (0,0,200,200);

Function name

Arguments in parentheses

Ends with semi-colon

fig. 2.4

MESSAGGI DI ERRORE

The screenshot shows the Processing IDE window titled "MyFirstProgram | Processing 0135 Beta". The code editor contains the following Pseudocode:

```
size(200,200);
background(255);
rectMode(CENTER);
stroke(0);
fill(150);
rect(100,100,20,100);

fill(255);
ellipse(100,70,60,60); ← Line 9 highlighted

fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);

stroke(0);
line(98,150,80,160);
line(118,150,120,160);
println("Take me to your leader!");
```

An arrow points from the text "Line 9 highlighted" to the line `ellipse(100,70,60,60);`. Another arrow points from the text "Error message" to the error message in the status bar: "No method named "ellipse" was found in type "Temporary_3082_2001". However, there is an accessible method "ellipse" whose name closely matches the name "ellipse".". A third arrow points from the text "Error message again!" to the same error message repeated in the status bar.

No method named "ellipse" was found in type "Temporary_3082_2001". However, there is an accessible method "ellipse" whose name closely matches the name "ellipse".

Error message again!

fig. 2.6

ESERCIZIO: TROVA L'ERRORE

Exercise 2-6: Fix the errors in the following code.



size(200,200); _____

background(); _____

stroke 255; _____

fill(150) _____

rectMode(center); _____

rect(100,100,50); _____



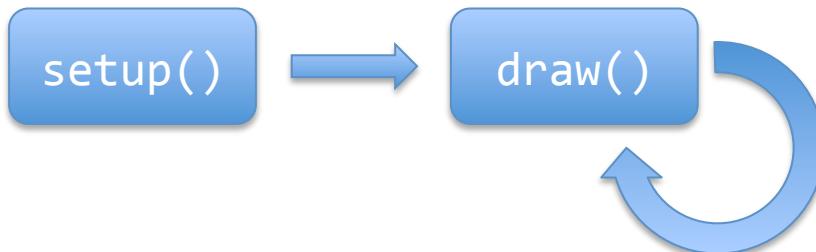
INTERACTION

FLUSSO DI UN PROGRAMMA PROCESSING

- Due fasi principali
 - SETUP
 - Questa parte viene eseguita all'inizio e ha lo scopo di configurare lo sketch per l'esecuzione
 - DRAW
 - Questa parte disegna la finestra ciclicamente, più volte al secondo. Aggiornando la visualizzazione si aggiorna anche lo stato del sistema
- Esempio: considera la corsa di una maratona
 - SETUP: indossa le scarpe e l'abbigliamento sportivo
 - DRAW: metti un piede davanti all'altro. Dopo 42km fermati.

FUNZIONI `setup()` E `draw()`

- `setup()` prepara lo spazio di visualizzazione e inizializza lo stato interno del programma. Viene eseguita solo una volta!
- `draw()` viene chiamata ripetutamente fino a quando il programma è in esecuzione. Ha il compito di aggiornare lo stato interno del programma e la visualizzazione
 - Sfruttiamo le continue esecuzioni di `draw()` per monitorare lo stato del sistema



BLOCCO DI CODICE

- Un **blocco** è un insieme di istruzioni racchiuse tra parentesi graffe {}
- Un blocco può contenere al suo interno altri blocchi
 - Di solito i blocchi annidati dentro altri sono pure indentati, ovvero hanno una rientranza nell'inizio della linea di codice
- Due blocchi di codice rilevanti sono **setup()** e **draw()**

FUNZIONI setup() E draw()

What's this?

What are these for?

```
void setup() {  
    // Initialization code goes here  
}
```

```
void draw() {  
    // Code that runs forever goes here  
}
```

fig. 3.1

FUNZIONI `setup()` E `draw()`

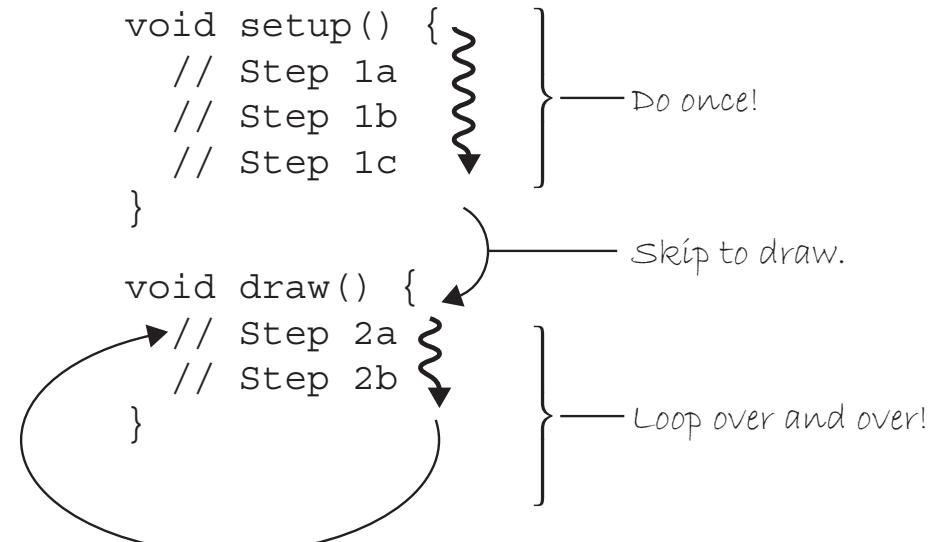


fig. 3.2

ESERCIZIO: ZOOG

```
void setup(){
    // Set the size of the window
    size(200,200);
}

void draw() {
    // Draw a white background
    background(255);

    // Set CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);

    // Draw Zoog's body
    stroke(0);
    fill(150);
    rect(100,100,20,100);

    // Draw Zoog's head
    stroke(0);
    fill(255);
    ellipse(100,70,60,60);

    // Draw Zoog's eyes
    fill(0);
    ellipse(81,70,16,32);
    ellipse(119,70,16,32);

    // Draw Zoog's legs
    stroke(0);
    line(90,150,80,160);
    line(110,150,120,160);
}
```

setup() runs first one time. **size()** should always be first line of **setup()** since Processing will not be able to do anything before the window size is specified.

draw() loops continuously until you close the sketch window.

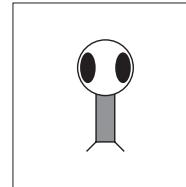


fig. 3.3

VARIAZIONI CON IL MOUSE

- Le variabili `mouseX` e `mouseY` sono due varibili di sistema.
- Sono aggiornate automaticamente per rappresentare la posizione orizzontale e verticale del puntatore del mouse sullo sketch
- Le variabili `pmouseX` e `pmouseY` sono aggiornate con la posizione del mouse alla iterazione precedente

VARIAZIONI CON IL MOUSE

Example 3-2: *mouseX* and *mouseY*

```
void setup() {  
    size(200,200);  
}  
  
void draw() {  
    background(255);  
  
    // Body  
    stroke(0);  
    fill(175);  
    rectMode(CENTER);  
    rect(mouseX,mouseY,50,50);  
}
```

Try moving **background()** to **setup()** and see the difference! (Exercise 3-3)

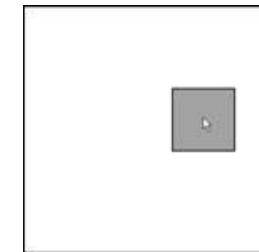


fig. 3.4

mouseX is a keyword that the sketch replaces with the horizontal position of the mouse.

mouseY is a keyword that the sketch replaces with the vertical position of the mouse.

Cosa accade se sposto l'istruzione **background(255)** in **setup()** ?