

# Corso di **Sistemi interattivi**

## **Lezione 3. Le basi della programmazione in Processing**

Prof. Rudy Melli ([rudymelli@ababrera.it](mailto:rudymelli@ababrera.it))

[www.vision-e.it/si](http://www.vision-e.it/si)

**ACCADEMIA DI BELLE ARTI DI BRERA**  
**Anno accademico 2018/2019**

---

# Programmazione incrementale: Approccio “un passo alla volta”

- ✓ Non partire subito dalla realizzazione di esempi complicati
  - ✓ Suddividere un problema in sottoparti (sottoproblemi) sempre meno complessi finché non appariranno semplici o ovvi ed iniziare a realizzarli uno alla volta
    - ✓ Se non si riesce a suddividere in sottoparti passare ad un problema meno complesso
  - ✓ Esempio “Space Invader” si può dividere in 3 sotto parti:
    - 1) Programmare la nave spaziale
    - 2) Programmare gli invasori
    - 3) Programmare il sistema di punteggio
  - ✓ Suddividiamo il problema 1 in sottoparti:
    - 1.1)Disegna un triangolo a schermo; il triangolo sarà la nostra nave spaziale
    - 1.2)Posiziona il triangolo in fondo allo schermo
    - 1.3)Sposta il triangolo leggermente a destra
    - 1.4)Anima il triangolo in modo che si muova da sinistra a destra
    - 1.5)Anima il triangolo da sinistra a destra solo quando il tasto Freccia destra è premuto
    - 1.6)Anima il triangolo da destra a sinistra solo quando il tasto Freccia sinistra è premuto
-

---

# Algoritmi

L'algoritmo è il mantra della programmazione, ma che cos'è?

Un algoritmo è una lista sequenziale di istruzioni che risolvono un particolare problema.

Immagina di spiegare a qualcuno di totalmente inesperto, istruzione per istruzione elementare, cosa deve fare per compiere un determinato compito, ecco, hai realizzato un algoritmo.

Ad esempio una ricetta da cucina è un algoritmo.

Provate a scrivere l'algoritmo: *Insegna a disegnare un muro di mattoni*

Suddividere in istruzioni banali (Es. 1. Disegna un rettangolo in basso a sinistra)

Suggerimenti:

- Fai cose differenti in base alle condizioni? Come usate le parole “se” e “altrimenti” nelle vostre istruzioni? (Esempio: se l'acqua è troppo fredda, aumenta l'acqua calda, altrimenti aumenta quella fredda)
- Usa la parola “ripeti” nelle tue istruzioni. Esempio: Disegna il rettangolo 5 volte

Parte 0)

Parte 1)

Parte 2)

Parte 3)

Parte 4)

Parte 5)

Parte 6)

Parte 7)

*NB: L'elenco parte con 0  
perché in programmazione si  
inizia a contare da 0  
(è bene abituarsi!)*

---

# Processing

Non sono tante le regole base della programmazione ed è importante impararle per poter capire come scrivere uno sketch da zero o anche come modificare un esempio.

## Maiuscole e minuscole

In Processing le lettere maiuscole e minuscole sono diverse quindi è necessario inserire correttamente i nomi dei comandi.

Ad esempio per disegnare una linea, **Line** è diverso da **line** ed il primo non è riconosciuto come comando!

## Codice

Il testo di uno sketch (il programma), viene chiamato codice perché è scritto con una sintassi guidata da regole definite dal linguaggio che si sta utilizzando. Ad esempio in Processing, il linguaggio è il Java

# Funzioni

Una funzione è un operazione (chiamata tecnicamente istruzione) che vogliamo che lo sketch (programma) esegua. Ad es. “Disegna una linea” è l’operazione che vogliamo fare, che si traduce nella funzione *line*.

Una funzione è identificata da un nome univoco (ad es. *line* identifica la funzione che disegna una linea) e può avere dei parametri che sono i suoi settaggi. Ad es. *line* necessita di 4 parametri x1,y1,x2,y2 che sono il punto iniziale e finale (in pixel).

La funzione completa si scrive con i parametri compresi tra parentesi tonde:

*line(10, 10, 30, 30)*

Non è finita! Infatti un’istruzione deve terminare con il carattere **;** per dire a Processing che abbiamo finito di dettagliare il comando, un po' come il punto alla fine di una frase.

Quindi l’istruzione completa è

*line(10, 10, 30, 30);*

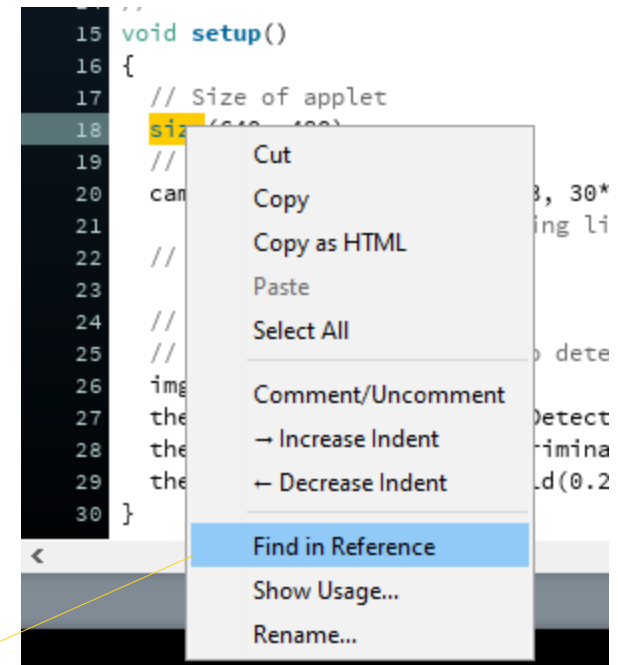
# Riferimenti ed Help

Esiste un elenco di tutte le funzioni base di Processing che è chiamato *Reference* ed è apribile dal menù Help->Reference

Oppure a questo indirizzo: <https://processing.org/reference/>

Selezionando il nome di una funzione nel codice e premendo il tasto destro del mouse selezionando *Find in Reference* si accede alla pagina con le istruzioni della funzione e la descrizione dei parametri.

Funziona anche con la combinazione di tasti CTRL-SHIFT-F oppure dal menù Help->Find in Reference



## Flusso dei dati

In uno sketch ci sono solitamente più funzioni, in fila, una sotto l'altra, utilizzate per fare operazioni più complesse.

Se ad esempio volessi disegnare un triangolo, lo potrei fare disegnando tre linee, quindi scrivendo 3 volte la funzione *line*:

```
line(10, 10, 30, 30);
```

```
line(30, 30, 10, 30);
```

```
line(10, 30, 10, 10);
```

Così ho detto a Processing di disegnare 3 linee!

NB: per disegnare un triangolo esiste in realtà la funzione

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

in cui inserire come parametri i 3 punti del triangolo!!!

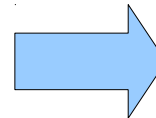
# Variabili

Le variabili sono parametri che memorizzano un valore da utilizzare nel programma e che può essere modificato dal programma stesso. Quando creiamo una variabile dobbiamo dichiarare: (1) **di che tipo è** (int, float, String, ...) (2) assegnarli un **nome** e (3) un **valore**. Ad esempio:

`int` contatore = 1;

In questo caso abbiamo creato una variabile di nome “contatore” di tipo int e con valore pari a 1, come ogni istruzione la riga finisce con il ;

```
1 |
2 int contatore = 1;
3 println("Valore di \"contatore\": " + contatore);
4 contatore = contatore + 1;
5 println("Valore di \"contatore\": " + contatore);
6 println(contatore);
7
```



```
Valore di "contatore": 1
Valore di "contatore": 2
2
```

Console Errors

Nell'esempio sopra variamo il valore della variabile “contatore” e scriviamo nella finestra di output (con la funzione *println*) il valore prima e dopo.



# Dati e tipi

In informatica i dati devono essere classificati per tipo per essere usati correttamente, ad esempio testo, numeri, immagini, ...

Più in specifico, per far capire a Processing con quale tipo di dato sta lavorando (eh sì, non è così intelligente da capirlo da solo!) esistono delle parole chiave che identificano questi tipi (attenzione sempre alle maiuscole ed alle minuscole):

- **void** Significa vuoto, è utilizzato solo per il tipo delle funzioni
- Numeri:
  - **int** abbreviazione di integer, intero, numeri senza virgola [ $-2^{31}$ ,  $-2^{31}-1$ ]
    - *Altri tipi di numeri senza virgola: **byte**, **long***
  - **float** Numeri reali, con virgola (32 bit)
    - **double** Numeri reali con virgola (64 bit)
- **boolean** Tipo logico, può assumere solo 2 valori *true*, *false* (vero, falso)
- **String** Testo, deve essere delimitato da doppio apice “
  - esempio “Questa è una stringa”
  - Per inserire il doppio apice in una stringa bisogna farlo precedere dal carattere speciale slash \, ad es. “Il suo nome è \"pippo\" ed è simpatico”
- **char** Carattere singolo, delimitato da apice singolo ‘ Ad es ‘Q’ ‘3’ ‘;’
- **PImage** Tipo immagine

# L'uso delle variabili

Le variabili possono essere dichiarate in qualsiasi punto del programma ma utilizzate solo nella parte di codice sotto la dichiarazione:

```
1
2 contatore = contatore + 1;
3 int contatore = 1;
4
5
```

The variable "contatore" does not exist

Viene segnalato un errore perché “contatore” è utilizzato prima (in termini di righe di codice) di

dichiarata.

Se dichiariamo una variabile fuori dalle funzioni, queste hanno una validità globale all'interno di tutte le funzioni create. Esempio:

```
1
2 void somma()
3 {
4     contatore = contatore + 1;
5 }
6 int contatore = 1;
7
```

Non viene segnalato nessun errore perché “contatore” è usata prima di essere dichiarata ma dentro ad una funzione!

## setup e draw

In Processing è possibile scrivere un programma semplice senza specificare un inizio ed una fine, ma esistono 2 funzioni fondamentali che si possono ritrovare nella maggior parte degli esempi:

### 1) **setup**

Al suo interno sono contenute tutte le operazioni preliminari che vengono eseguite solo all'inizio. Ad esempio la creazione della finestra di rendering e/o l'apertura della webcam, ecc.

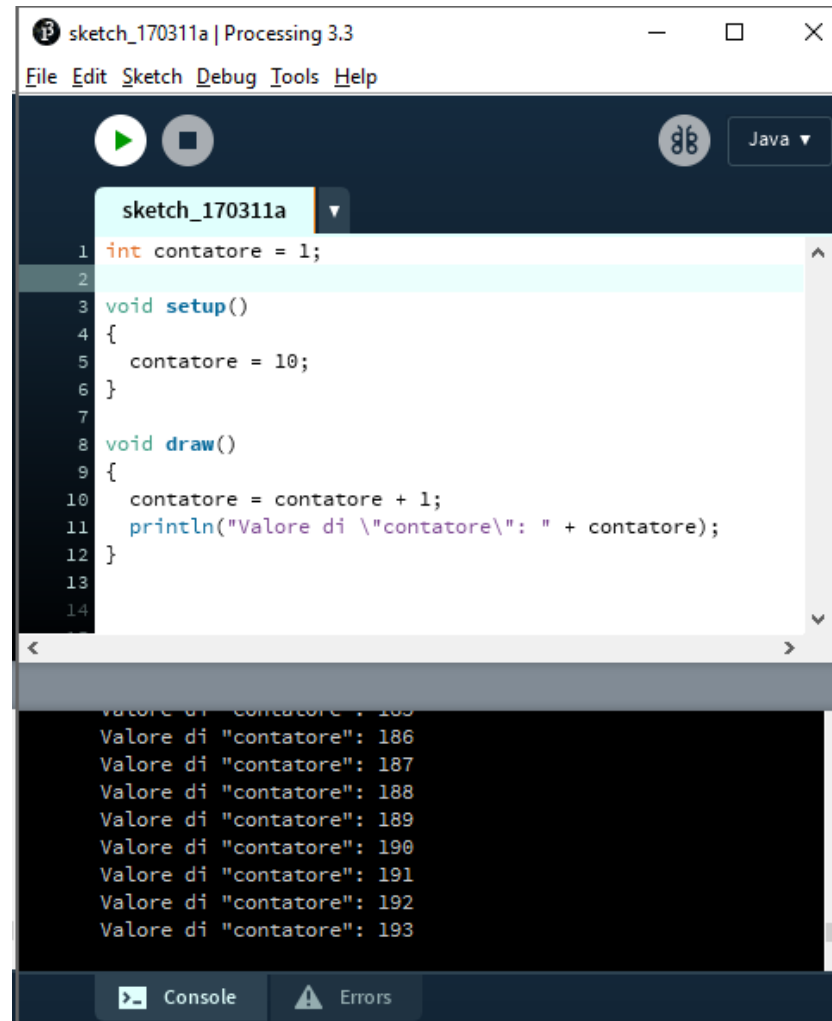
Questa è la funzione “chiamata”, cioè eseguita, per prima

### 2) **draw**

E' una funzione molto particolare perché viene ripetuta ad ogni istante di rendering, cioè ad ogni frame, al suo interno sono contenute le operazioni di lettura di sensori e/o calcolo di operazioni e/o analisi e/o output (disegno)

# setup e draw

In questo esempio la variabile contatore viene incrementata di 1 ad ogni ciclo di rendering



```
sketch_170311a | Processing 3.3
File Edit Sketch Debug Tools Help

sketch_170311a
1 int contatore = 1;
2
3 void setup()
4 {
5   contatore = 10;
6 }
7
8 void draw()
9 {
10  contatore = contatore + 1;
11  println("Valore di \"contatore\": \" + contatore);
12 }
13
14

Valore di \"contatore\": 186
Valore di \"contatore\": 187
Valore di \"contatore\": 188
Valore di \"contatore\": 189
Valore di \"contatore\": 190
Valore di \"contatore\": 191
Valore di \"contatore\": 192
Valore di \"contatore\": 193

Console Errors
```

# Funzione personalizzata

E' possibile scrivere una funzione personalizzata.....ma un neofita potrebbe pensare, a cosa può servire?!?!

## 1) **Semplificazione**

Ripetere una serie di operazioni identiche (o con valori di parametri differenti) più volte senza riscrivere tutto il codice. Ad esempio potrei creare una funzione che disegna una stella e poi ripeterla più volte

## 2) **Riusabilità**

La funzione può essere riutilizzata in altri progetti

## 3) **Leggibilità**

Una funzione può fungere anche da contenitore per raccogliere parti di programma che racchiudono una macro operazione

Le funzioni iniziano sempre con il tipo di dato da restituire (void, int, float, ...) seguito da uno spazio e dal nome della funzione che deve iniziare con una lettera (minuscola o maiuscola) e non può contenere alcuni caratteri speciali(spazio, punto, punto e virgola, ....)


## Parametri di una funzione

Sono i valori che scriviamo tra parentesi tonda e che vengono inviate alla funzione per renderla personalizzata. Sono opzionali.

Le funzioni iniziano sempre con la parentesi graffa aperta { e finiscono con la parentesi graffa chiusa }

Ad esempio. Funzione matematica che calcola la media tra 2 numeri:

```
4
5 float media(float a, float b)
6 {
7     return (a+b)/2;
8 }
9
10 void setup()
11 {
12     float c = media(5, 10);
13     println(c);
14 }
```



NB: la funzione setup è necessaria quando si lavora con le funzioni. Dentro andranno scritte le operazioni da fare.

# Identazione

Quando si scrive una funzione o un ciclo ed in ogni caso ogni volta sia presente un blocco di dati che inizia con **{** e finisce con **}**, è buona norma indentare il codice contenuto, cioè iniziare a scrivere un po' più a destra spostandosi con uno step di TAB. Ad esempio:

```
void triangle()      OK  
{  
  line(10, 10, 20, 20);  
  line(20, 20, 10, 20);  
  line(10, 20, 10, 10);  
}
```

```
void triangle()      NO  
{  
  line(10, 10, 20, 20);  
  line(20, 20, 10, 20);  
  line(10, 20, 10, 10);  
}
```

Il caso di destra non è un errore di sintassi quindi lo sketch funzionerà comunque ma la sua leggibilità sarà decisamente inferiore.

# Condizioni

Le condizioni permettono di controllare il funzionamento del programma ed implementare delle scelte che ne cambiano il comportamento.

Esempi di condizioni:

*Se  $X > 100$  disegna una linea rossa, altrimenti nera*

*Se  $X, Y$  sono contenuti dentro una zona riproduci un suono*

Esistono principalmente 2 tipi di istruzioni condizionali: *if* e *switch*

Le condizioni vengono solitamente utilizzate insieme agli operatori relazionali:

- ◆ `==` uguale a
- ◆ `!=` diverso da
- ◆ `>` maggiore di
- ◆ `<` minore di
- ◆ `>=` maggiore o uguale a
- ◆ `<=` minore o uguale a



## Condizione *if*

Questa è la condizione base, la più utilizzata il cui funzionamento dipende dal nome *if* (in italiano *se*) e la sua sintassi è la seguente:

```
if(test)
{
    codice
}
```

Cioè se è valida la condizione *test* allora esegue il *codice* altrimenti no.

La condizione può essere composta da più condizioni contemporanee utilizzando le parole chiave `&&` (and, e) o `||` (or, oppure).

Ad esempio controllare che il valore della variabile *x* sia tra 10 e 20:

```
if(x >= 10 && x <= 20) { }
```

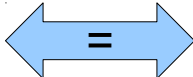
La parola chiave opzionale *else* (altrimenti) estende la funzionalità dell'*if*, e viene utilizzata dopo la parentesi graffa chiusa per eseguire una porzione di codice quando la condizione *test* NON è vera.

```
if(x >= 10 && x <= 20)
{
}
else
{
}
```

## Condizione *if*

Se il codice da eseguire è composto da un'unica istruzione le parentesi graffe sono opzionali:

```
if(x >= 10 && x <= 20)
  fill(0);
else
  fill(255);
```



```
if(x >= 10 && x <= 20)
{
  fill(0);
}
else
{
  fill(255);
}
```

E' possibile inserire un'altra condizione *if* in cascata dopo l'*else*

```
if(x >= 10 && x <= 20)
  fill(0);
else if(x >= 11 && x <= 30)
  fill(50);
else if(x >= 21 && x <= 40)
  fill(100);
else
  fill(255);
```

Non esiste un limite al numero di condizioni in cascata possibili.

## Condizione *switch*

E' un costrutto più complicato che serve per semplificare il codice quando dobbiamo verificare la corrispondenza di una variabile ad un valore specifico. La sintassi è

```
switch (var)
{
    case valore1: codice1; break;
    case valore2: codice2; break;
    case valore3: codice3; break;
    default: codice4; break;
}
```

*var* è la variabile da controllare

valore1, ... sono i valori che può assumere *var*

Verrà eseguito solo il codice corrispondente al valore corrente di *var*, se non vi è corrispondenze verrà eseguito il codice della condizione *default*, se presente

# Ripetere operazioni

Spesso capita di dover ripetere un'operazione più volte, o in modo identico, o cambiando alcuni parametri. Un esempio potrebbe essere voler disegnare un cerchio 10 volte con diametro e colori diversi scelti casualmente con la funzione `random`.

Una soluzione senza usare un ciclo è questa (22 righe di codice) in cui `fill` (colore) ed `ellipse` (disegna un ellisse) vengono copiati ed incollati 10 volte:

```
1 size(500, 500);
2 background(0);
3 fill(random(255), random(255), random(255));
4 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
5 fill(random(255), random(255), random(255));
6 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
7 fill(random(255), random(255), random(255));
8 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
9 fill(random(255), random(255), random(255));
10 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
11 fill(random(255), random(255), random(255));
12 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
13 fill(random(255), random(255), random(255));
14 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
15 fill(random(255), random(255), random(255));
16 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
17 fill(random(255), random(255), random(255));
18 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
19 fill(random(255), random(255), random(255));
20 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
21 fill(random(255), random(255), random(255));
22 ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
```



# Ciclo

Esistono dei costrutti chiamati **cicli** che permettono di ripetere operazioni con poche righe di codice e con la possibilità di variare i parametri.

Vediamo l'esempio precedente usando un ciclo:

```
1 size(500, 500);  
2 background(0);  
3 for(int i=0; i<10; ++i)  
4 {  
5   fill(random(255), random(255), random(255));  
6   ellipse(random(400) + 50, random(400) + 50, random(100), random(100));  
7 }
```

Il risultato è esattamente lo stesso con 7 righe di codice!

In questo caso le 2 righe

```
fill(random(255), random(255), random(255));  
ellipse(random(400) + 50, random(400) + 50, random(100), random(100));
```

Vengono ripetute 10 volte grazie al ciclo *for*

Esistono 2 tipi di cicli: *for* e *while*

NB: `++i` è come scrivere `i=i+1`

## Ciclo for

Il ciclo *for* inizia la parola chiave **for** e all'interno delle parentesi tonde vi sono 3 parti (inizializzazione; test; aggiornamento) separate dal carattere **;**

- 1) Viene fatta l'inizializzazione, solo una volta
- 2) Se la condizione *test* è verificata, cioè è vera, il codice contenuto dentro le parentesi graffe viene ripetuto
- 3) Se la condizione *test* NON è verificata, cioè è false, il ciclo finisce e si riprende l'esecuzione da dopo la parentesi graffa chiusa
- 4) Viene eseguito l'*aggiornamento*
- 5) Ripetere da 2

Ad esempio:

```
for(int i=0; i<10; i=i+1)
{
  ...
}
```

Inizializzazione: *int i=0* → Dichiarazione di una variabile di tipo *int* e valore 0

Test: *i<10* → i è minore di 10?

Aggiornamento: *i=i+1* → Incrementa i di 1

## Ciclo for

Provare questo esempio in uno sketch:

```
for (int i = 0; i < 80; i = i+5)
{
  line(30, i, 80, i);
}
```

## Ciclo while

Il ciclo *while* inizia la parola chiave **while** che all'interno delle parentesi tonde si aspetta un *test*. Finchè *test* è verificato (cioè vero) il codice dentro le parentesi graffe viene ripetuto.

Esempio:

```
int i = 0;
while (i < 80)
{
    line(30, i, 80, i);
    i = i + 5;
}
```



## Esercizi

- 1) Creare un programma che crea una variabile e disegna una linea rossa se questo valore è  $\geq 0$  e nera se è  $< 0$
- 2) Creare un programma che disegna 80 linee verticali nere ogni 5 pixel a partire da  $x=0$
- 3) Come 2, ma le prime 40 linee devono essere disegnate nere e le successive rosse
- 4) Usare le coordinate del mouse (*mouseX*, *mouseY*). Cambiare il colore dello sfondo (*background()*) in funzione della coordinata X del mouse, quando *MouseX* è maggiore della metà della larghezza della finestra (*width*) lo sfondo deve essere nero, altrimenti verde.
- 5) Testare gli esempi
  - 1) *Basics/Color*
  - 2) *Basics/Input*
  - 3) *Basics/Math*
  - 4) *Basics/Web*