

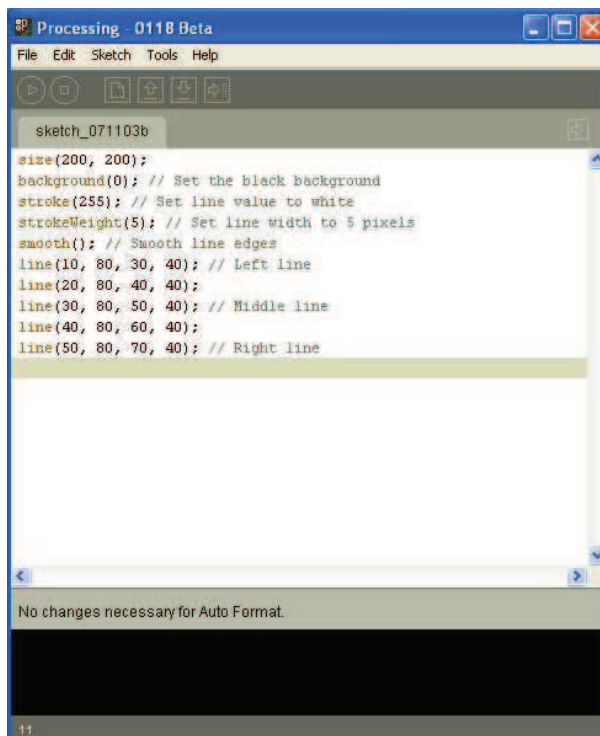
L'ambiente di programmazione PROCESSING

Processing è un ambiente di programmazione sviluppato al Media Lab del MIT, il cui scopo è di permettere ad artisti, designer ed in generale a persone con scarse cognizioni di informatica ma attratte dalle possibilità di applicazione dell'informatica all'arte e al design di poter scrivere dei programmi, più o meno sofisticati, in breve tempo, utilizzando uno strumento facile da usare ma sufficientemente potente per fare delle sperimentazioni nel settore della "computer art" e dell' "interaction design".

Processing è un prodotto open source disponibile per le piattaforme Windows, Macintosh e Linux; lo si può scaricare dal [sito ufficiale](#).

Potete scaricare il sommario, l'indice analitico, alcuni capitoli e gli esempi di programmazione del [libro pubblicato dagli autori di processing](#)

Processing è molto semplice da usare: si scrive il testo di un programma e lo si esegue (con il comando "run" o premendo il pulsante "Play" all'estrema sinistra della finestra dell'applicazione). I programmi scritti possono ovviamente essere salvati e successivamente modificati. Il tutto si fa in modo semplice e intuitivo, così che anche un utente inesperto possa imparare in pochi minuti ad usarlo.



Come abbiamo detto, Processing è un ambiente di programmazione, ovvero un pacchetto software per scrivere programmi.

Il linguaggio in cui tali programmi devono essere scritti è chiamato Processing (quindi attenzione: ambiente di programmazione e linguaggio hanno lo stesso nome) ed è basato sul linguaggio di programmazione Java.

Infatti la sintassi di Processing ricorda molto quella di Java, ma è molto più semplice scrivere un programma con Processing che un programma Java, anche se quest'ultimo ha una maggiore flessibilità (e quindi complessità) ed è un linguaggio di applicabilità generale, basato sul paradigma di programmazione a oggetti.

In effetti, il linguaggio Processing è diviso in tre livelli di complessità: elementare, intermedio, avanzato. Quest'ultimo livello coincide sostanzialmente con il linguaggio Java stesso. La maggior parte degli esempi di programmazione che vedremo ricadono nei due primi livelli.

Primi esempi di programmazione

Abbiamo visto in una lezione precedente che un programma lo si scrive partendo da istruzioni elementari e combinandole in tre modi possibili, usando opportune strutture di controllo.

Le strutture di controllo di un linguaggio di programmazione sono dei costrutti (frasi) del linguaggio che controllano l'ordine di esecuzione delle istruzioni di un programma.

- **Sequenza**: le istruzioni sono eseguite in sequenza una dopo l'altra.
- **Selezione**: scelta dell'istruzione da eseguire tra più alternative.
- **Iterazione**: ripetizione (un certo numero di volte) di un gruppo di istruzioni.

Sequenza

Sequenza : giustapposizione di istruzioni separate da ";" (punto e virgola)

Una o più istruzioni possono essere raggruppate tra parentesi graffe

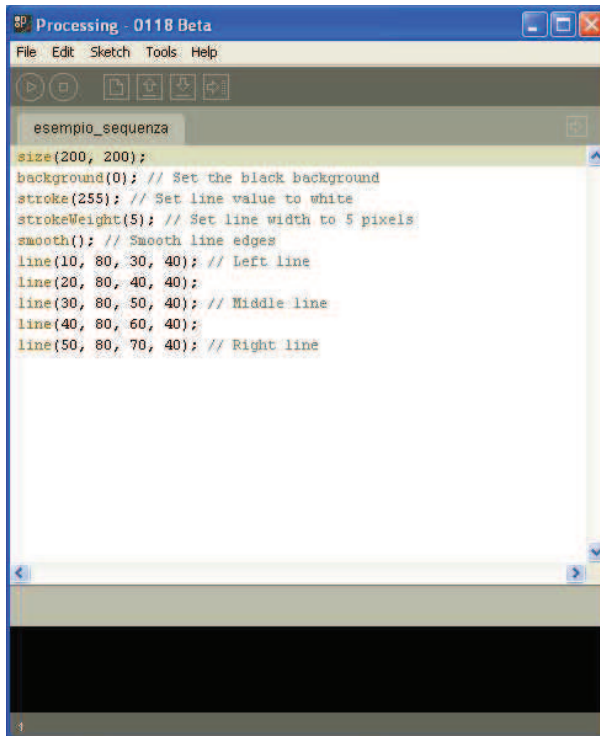
```
{I1; ...; In;
```

per formare una singola istruzione. Le istruzioni sono eseguite nell'ordine in cui appaiono:

```
I1, ... , In
```

Esempio

Il primo esempio di programma Processing disegna cinque rette oblique tra loro parallele.



```

Processing - 0118 Beta
File Edit Sketch Tools Help

esempio_sequenza

size(200, 200);
background(0); // Set the black background
stroke(255); // Set line value to white
strokeWeight(5); // Set line width to 5 pixels
smooth(); // Smooth line edges
line(10, 80, 30, 40); // Left line
line(20, 80, 40, 40);
line(30, 80, 50, 40); // Middle line
line(40, 80, 60, 40);
line(50, 80, 70, 40); // Right line

```

l'istruzione `size(200, 200)` crea una finestra di 200 per 200 pixel entro cui avviene il disegno,

l'istruzione `background(0)` sceglie il nero come colore dello sfondo,

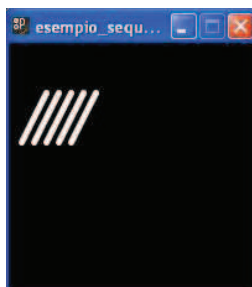
l'istruzione `stroke(255)` sceglie il bianco come colore per il disegno delle linee,

l'istruzione `strokeWeight(5)` imposta lo spessore delle linee a 5 pixel,

l'istruzione `smooth()` arrotonda l'estremità delle linee.

Ciascuna delle 5 linee di codice che seguono disegna una linea, ad es. l'istruzione `line(10, 80, 30, 40)` disegna una linea con estremi nei punti di coordinate (10, 80) e (30, 40).

Il sistema di coordinate di Processing pone l'origine degli assi nell'angolo superiore sinistro della finestra di disegno, per cui spostandosi verso destra aumenta il valore della prima coordinata di un punto, mentre spostandosi verso il basso aumenta il valore della seconda coordinata.



Si noti che l'immagine appare in bianco e nero, poiché è stato scelto di disegnare con immagini a livelli di grigio, scelti tra 256 tonalità diverse: 0 per il nero e 255 per il bianco.

Ogni istruzione del programma è seguita da un **commento**, un testo inserito come documentazione del programma, che non viene considerato come codice, ma viene scartato durante l'esecuzione del programma. I commenti usati in questo programma sono **monoriga**,

ovvero iniziano dopo i simboli "/" e terminano in fondo alla riga.

Uso delle variabili

Se si osserva attentamente il programma precedente, le cinque righe parallele sono disegnate incrementando di 10 unità la prima componente dei punti estremi di ogni retta.

Il prossimo esempio differisce dal precedente per l'introduzione di una variabile intera di nome *x* a cui si assegna inizialmente il valore 0 con la **dichiarazione**

```
int x = 0
```

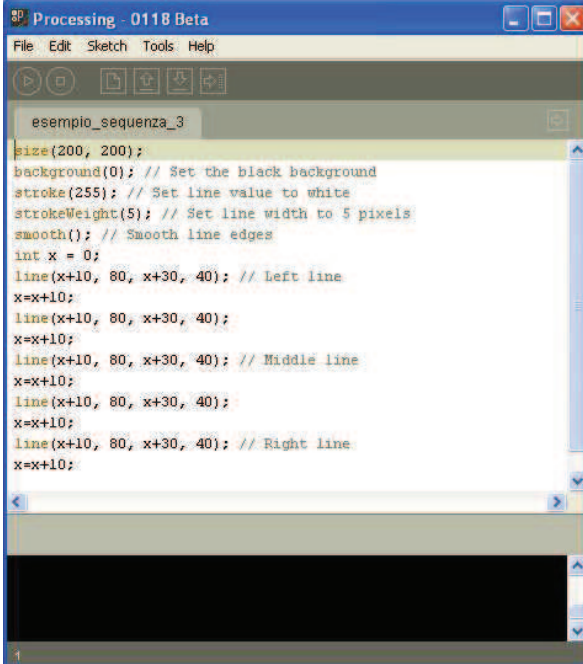
e dopo il disegno di ogni linea è incrementata di 10 unità con il **comando di assegnamento**

```
x=x+10 .
```

Una dichiarazione ha il compito di creare una variabile stabilendo il nome e l'insieme dei valori che la variabile può assumere (ovvero il tipo della variabile, in questo caso di tipo *int* che denota i numeri interi), nonché di riservare nella memoria del calcolatore lo spazio necessario per memorizzare il valore corrente della variabile.

In Processing come in Java e tanti altri linguaggi, una variabile non può essere usata se non è prima dichiarata. Invece vedremo che in JavaScript la dichiarazione di una variabile non è necessaria.

L'effetto finale del programma è sempre lo stesso, ma, a differenza del programma precedente, il codice mette bene in evidenza la relazione tra le coordinate delle 5 linee.



```
Processing - 0118 Beta
File Edit Sketch Tools Help
esempio_sequenza_3
size(200, 200);
background(0); // Set the black background
stroke(255); // Set line value to white
strokeWeight(5); // Set line width to 5 pixels
smooth(); // Smooth line edges
int x = 0;
line(x+10, 80, x+30, 40); // Left line
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40); // Middle line
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40); // Right line
x=x+10;
```

Iterazione

I costrutti di iterazione eseguono un certo numero di volte un'istruzione, detta **corpo**. Un

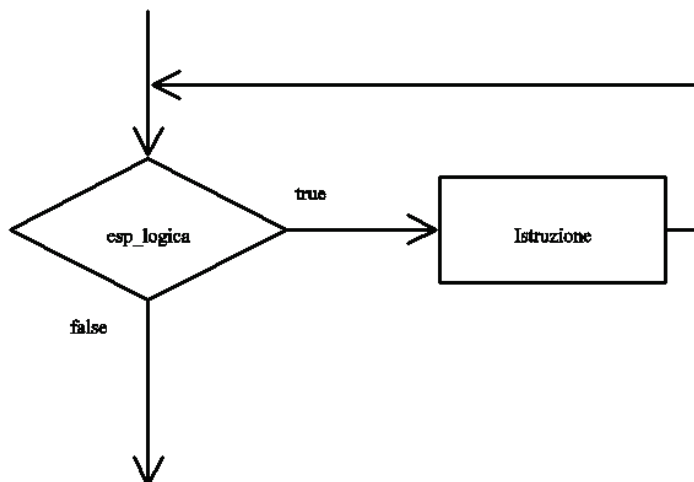
costrutto di iterazione è anche detto **istruzione di ciclo**, o semplicemente **ciclo**.

Noi esamineremo due tipi di costrutti di iterazione:

comando while

```
while (<espressione logica> ) I
```

L'esecuzione di un comando while causa l'esecuzione dell'istruzione I (il **corpo** del while) se l'espressione logica (la **guardia** del while) è vera, poi si valuta di nuovo l'espressione logica: se è vera si esegue di nuovo l'istruzione, e così via finché l'espressione logica diventa falsa.

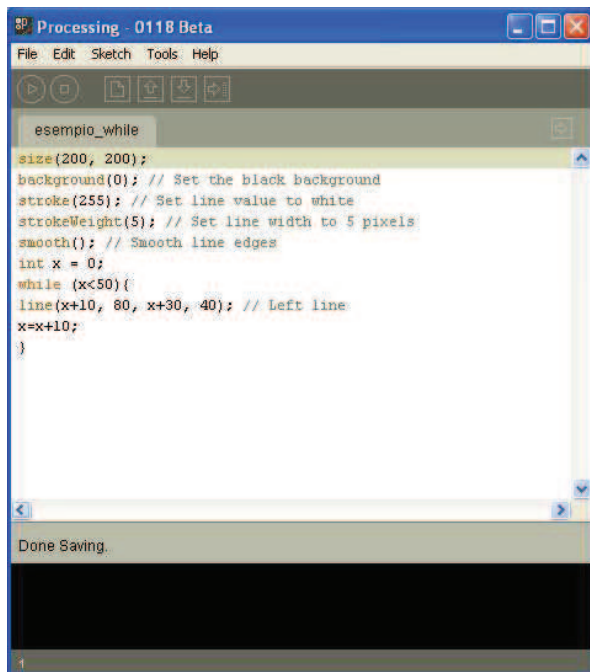


Esempio di uso del comando while

Osservando il codice degli esempi precedenti, le cinque righe parallele sono disegnate incrementando di 10 unità la prima componente dei punti estremi di ogni retta.

Si ha quindi una ripetizione (5 volte) dell'esecuzione del comando di tracciatura delle linee.

La tracciatura delle linee può avvenire senza scrivere 5 volte lo stesso comando, usando il comando while.

The image shows a screenshot of the Processing IDE window titled "Processing - 0118 Beta". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu is a toolbar with icons for running, saving, and other functions. The main area is a code editor with a file named "esempio_while". The code in the editor is as follows:

```
size(200, 200);
background(0); // Set the black background
stroke(255); // Set line value to white
strokeWeight(5); // Set line width to 5 pixels
smooth(); // Smooth line edges
int x = 0;
while (x<50){
  line(x+10, 80, x+30, 40); // Left line
  x=x+10;
}
```

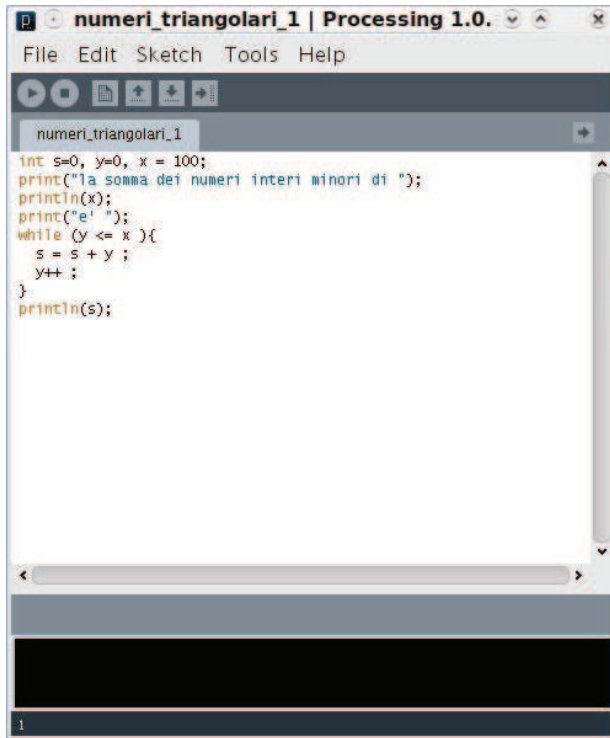
At the bottom of the IDE, there is a status bar that says "Done Saving." and a small black preview window.

Il corpo del ciclo while disegna una linea ed incrementa il valore della variabile intera x di 10 unità.

Pertanto, la variabile x assume i valori 0, 10, 20, 30, 40, in corrispondenza dei quali una linea viene disegnata. Quando x assume il valore 50, la guardia del ciclo while diventa falsa e

l'esecuzione del ciclo termina.

La figura seguente mostra come il ciclo while possa essere usato per calcolare i numeri triangolari, ovvero i numeri interi positivi ottenuti come somma dei numeri interi compresi tra 1 e un intero positivo n .



```
numeri_triangolari_1
int s=0, y=0, x = 100;
print("la somma dei numeri interi minori di ");
println(x);
print("e' ");
while (y <= x ){
  s = s + y ;
  y++ ;
}
println(s);
```

Il programma funziona in modo analogo al diagramma a blocchi discusso in una lezione precedente (teoria degli algoritmi).

comando for

for (I_1 ; E ; I_2) I_3

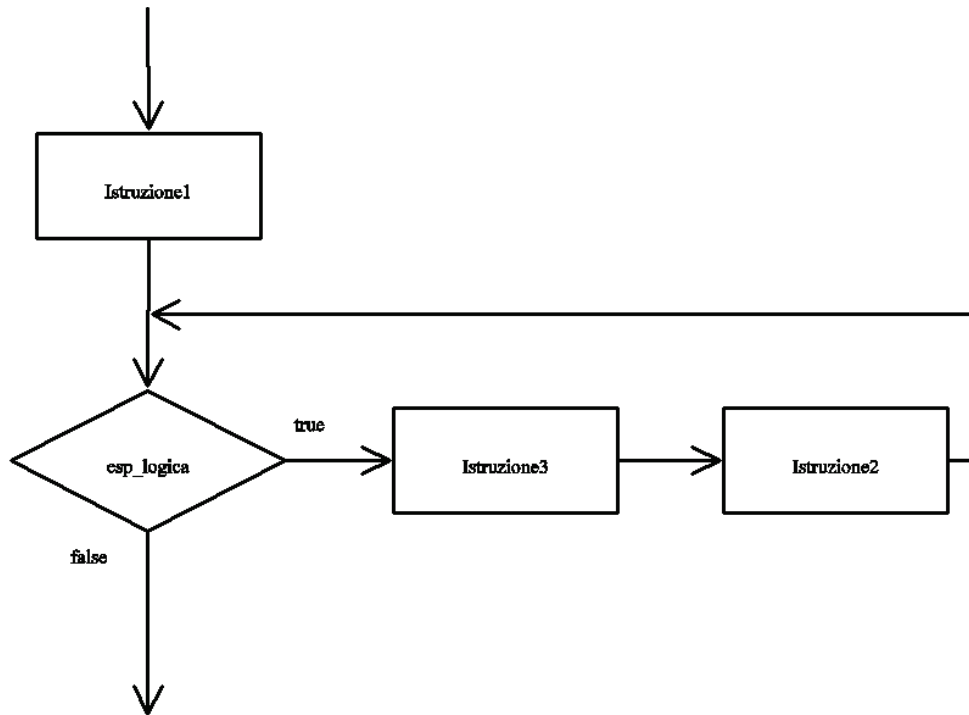
I_1 è l'istruzione di inizializzazione;

E è la guardia del for ;

I_2 è l'istruzione di incremento o aggiornamento;

I_3 è il corpo del for ;

l'esecuzione di un comando for avviene come segue: innanzi tutto è eseguita l'istruzione I_1 di inizializzazione (che di solito assegna un valore ad una variabile, detta variabile di controllo del ciclo for); se la guardia è vera si esegue il corpo I_3 e l'aggiornamento I_2 , poi si valuta di nuovo la guardia: se è vera si esegue di nuovo il corpo e l'aggiornamento, finché la guardia diventa falsa.

**Esempio di uso del comando for**

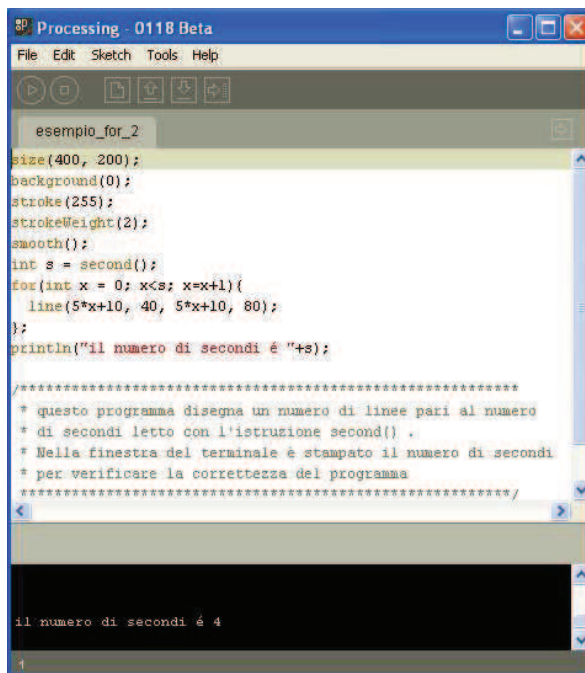
Questo esempio è una riscrittura del programma precedente usando il ciclo for. Si noti che

l'istruzione che assegna 0 alla variabile x è l'istruzione di inizializzazione del ciclo for, mentre l'istruzione che incrementa di 10 unità il valore di x è l'istruzione di aggiornamento del ciclo for. Il corpo del for è costituito dalla sola istruzione di tracciatura di una linea e viene ripetuto 5 volte.

```
Processing - 0118 Beta
File Edit Sketch Tools Help
esempio_for
size(200, 200);
background(0); // Set the black background
stroke(255); // Set line value to white
strokeWeight(5); // Set line width to 5 pixels
smooth(); // Smooth line edges
for(int x = 0; x<50; x=x+10){
  line(x+10, 80, x+30, 40);
}
```

Attenzione: le istruzioni di ciclo non sono solo dei semplici sostituti del comando di sequenzializzazione che permettono di abbreviare i programmi, ma permettono di fare molto di più. Se non usiamo i cicli e vogliamo disegnare 10 righe anziché 5, dobbiamo modificare il programma, aggiungendo 5 nuovi comandi di disegno.

Il prossimo esempio mostra come con le istruzioni di iterazione sia possibile disegnare un numero di linee che dipende dal numero di secondi del tempo in cui si lancia l'esecuzione del programma: non è pertanto stabilire a priori quante linee si devono disegnare e un tale programma non si può realizzare con la sola sequenzializzazione di comandi.

The image shows a screenshot of the Processing IDE window titled "Processing - 0118 Beta". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar are icons for running, stopping, and saving. The code editor shows the following code:

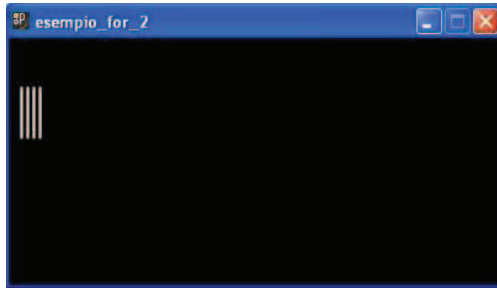
```
esempio_for_2
size(400, 200);
background(0);
stroke(255);
strokeWeight(2);
smooth();
int s = second();
for(int x = 0; x<s; x=x+1){
  line(5*x+10, 40, 5*x+10, 80);
};
println("il numero di secondi è "+s);

/*****
 * questo programma disegna un numero di linee pari al numero
 * di secondi letto con l'istruzione second() .
 * Nella finestra del terminale è stampato il numero di secondi
 * per verificare la correttezza del programma
 *****/
```

The terminal window at the bottom shows the output: "il numero di secondi è 4".

Alla variabile intera `s` è assegnato il numero dei secondi letti dall'orologio del calcolatore.

Nell'immagine precedente il testo che appare nella finestra del terminale indica che il valore assegnato ad `s` è 4. Il programma disegnerà pertanto quattro righe, come mostrato nella figura seguente:



Il corpo

```
line(5*x+10, 40, 5*x+10, 80);
```

del comando for successivo che esegue la tracciatura di una linea verticale è eseguito un numero di volte pari al valore della variabile *s*: infatti, la variabile *x* di conteggio del for è inizializzata a 0 e fintanto che il suo valore è minore di quello di *s*, si traccia una linea e si incrementa il valore di *x*.

In generale, se *s* è una variabile intera con un valore definito, l'esecuzione del ciclo

```
for(int x = 0; x<s; x=x+1){
    /* corpo del for */
};
```

provoca l'esecuzione ripetuta *s* volte del corpo del for.

Con le istruzioni di ciclo, si può incappare nel cosiddetto "fencepost error" (errore dello steccato e dei pali), che consiste nell'eseguire il corpo del ciclo una volta in più o in meno del numero corretto di iterazioni. Per essere sicuri che il programma funzioni correttamente, provate ad assegnare dei valori piccoli alla variabile *s*, ad es. 0 o 1 e vedete cosa succede: se

le prove hanno successo, allora il programma **dovrebbe** essere corretto.

Per concludere l'analisi di quest'esempio, si noti il **commento multiriga** in fondo al codice. Si tratta di un tipo di commento che inizia con i simboli `/*`, termina con i simboli `*/` e può occupare un numero arbitrario di righe di testo.

Selezione

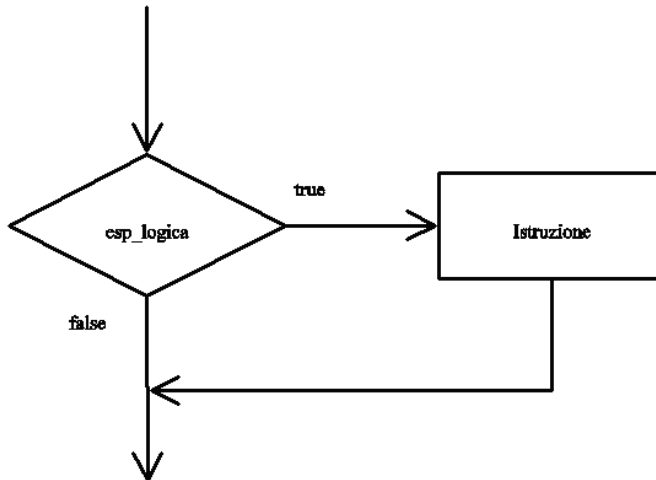
I costrutti di selezione scelgono di eseguire un'istruzione piuttosto che un'altra in base al valore di un'espressione logica (espressione che assume solo due valori, **vero** o **falso**, e pertanto una tale espressione è anche detta **espressione booleana**, da George Boole, matematico e logico del diciannovesimo secolo).

Noi esamineremo due tipi di costrutti di selezione:

comando if

```
if (<espressione logica> ) I1
```

L'esecuzione di tale comando causa l'esecuzione dell'istruzione *I*₁ se l'espressione logica (detta **guardia**) è vera



Esempio di uso del comando if

Il seguente programma disegna 5 righe parallele di colore grigio se, al momento dell'esecuzione del programma, l'orologio del calcolatore indica un'orario in cui il valore dei secondi è inferiore a 30; in caso contrario, le righe sono di colore bianco. La decisione di quale colore applicare è fatta con il costrutto if.

```
Processing - 0118 Beta
File Edit Sketch Tools Help

esempio_if$

size(200, 200);
background(0); // Set the black background
stroke(255); // Set line value to white
strokeWeight(5); // Set line width to 5 pixels
smooth(); // Smooth line edges
int x = 0;
int s = second(); // read seconds
if (s<30){
  stroke(128); // Set line value to gray
};
print("secondi = " + s);
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
Done Saving.

28
```

Il colore delle linee da tracciare viene preimpostato con la terza riga di codice

```
stroke(255)
```

che sceglie il colore bianco. Il successivo comando if

```
if (s<30){  
    stroke(128); // Set line value to gray  
};
```

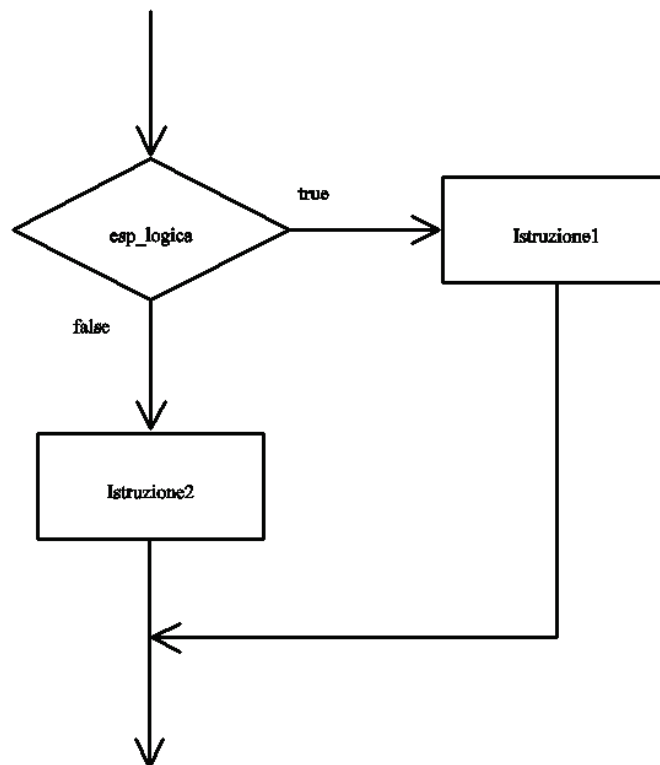
imposta con l'istruzione `stroke(128)` il colore grigio come colore di tracciatura se il numero dei secondi, contenente nella variabile `s` è minore di 30.

comando if-else

```
if (<espressione logica> ) I1
```

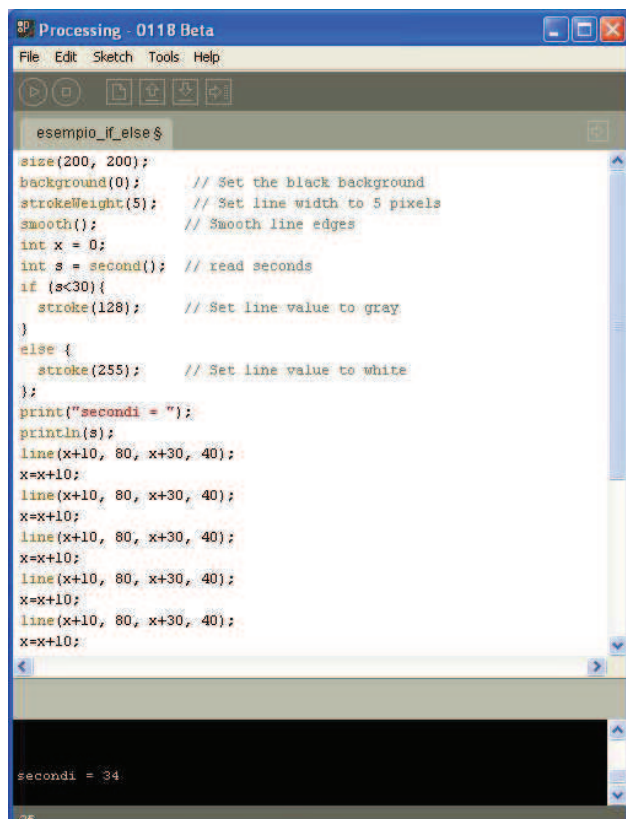
```
else I2
```

L'esecuzione di tale comando causa l'esecuzione dell'istruzione I_1 se l'espressione logica è vera; dell'istruzione I_2 in caso contrario.



Esempio di uso del comando if-else

Il seguente programma disegna 5 righe parallele di colore grigio se, al momento dell'esecuzione del programma, l'orologio del calcolatore indica un'orario in cui il valore dei secondi è inferiore a 30; in caso contrario, le righe sono di colore bianco. La decisione di quale colore applicare è fatta con il costrutto if-else

The image shows a screenshot of the Processing IDE window titled "Processing - 0118 Beta". The window has a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for running, saving, and other functions. The main area is a code editor with the following code:

```
esempio_if_else $
size(200, 200);
background(0); // Set the black background
strokeWeight(5); // Set line width to 5 pixels
smooth(); // Smooth line edges
int x = 0;
int s = second(); // read seconds
if (s<30){
  stroke(128); // Set line value to gray
}
else {
  stroke(255); // Set line value to white
};
println("secondi = ");
println(s);
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
line(x+10, 80, x+30, 40);
x=x+10;
```

At the bottom of the window, there is a console window showing the output:

```
secondi = 34
```

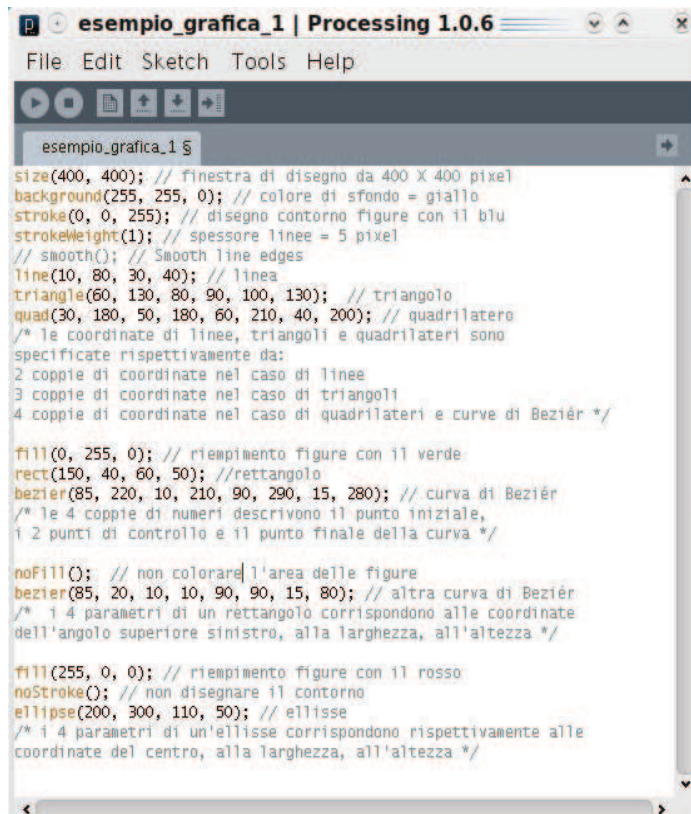
A differenza dell'esempio precedente, il colore delle linee da tracciare non viene preimpostato, ma viene determinato dal comando if-else

```
if (s<30){  
    stroke(128); // Set line value to gray  
}  
else {  
    stroke(255); // Set line value to white  
};
```

che imposta con l'istruzione `stroke(128)` il colore grigio come colore di tracciatura se il numero dei secondi, contenente nella variabile `s` è minore di 30, mentre in caso contrario il colore di tracciatura è impostato al colore bianco con l'istruzione `stroke(255)`.

Primo esempio di grafica 2D

Il prossimo semplicissimo programma mostra come disegnare semplici figure geometriche. Per un approfondimento, si veda il link [GRAFICA 2D](#) e per informazioni sull'uso delle primitive grafiche 2D (istruzioni per il disegno di figure geometriche bidimensionali, si consulti l'elenco denominato "2D Primitives" alla pagina del [manuale di riferimento di Processing](#) .



```
esempio_grafica_1 | Processing 1.0.6  
File Edit Sketch Tools Help  
esempio_grafica_1 §  
size(400, 400); // finestra di disegno da 400 X 400 pixel  
background(255, 255, 0); // colore di sfondo = giallo  
stroke(0, 0, 255); // disegno contorno figure con il blu  
strokeWeight(1); // spessore linee = 5 pixel  
// smooth(); // Smooth line edges  
line(10, 80, 30, 40); // linea  
triangle(60, 130, 80, 90, 100, 130); // triangolo  
quad(30, 180, 50, 180, 60, 210, 40, 200); // quadrilatero  
/* le coordinate di linee, triangoli e quadrilateri sono  
specificate rispettivamente da:  
2 coppie di coordinate nel caso di linee  
3 coppie di coordinate nel caso di triangoli  
4 coppie di coordinate nel caso di quadrilateri e curve di Beziér */  
  
fill(0, 255, 0); // riempimento figure con il verde  
rect(150, 40, 60, 50); //rettangolo  
bezier(85, 220, 10, 210, 90, 290, 15, 280); // curva di Beziér  
/* le 4 coppie di numeri descrivono il punto iniziale,  
1 2 punti di controllo e il punto finale della curva */  
  
noFill(); // non colorare l'area delle figure  
bezier(85, 20, 10, 10, 90, 90, 15, 80); // altra curva di Beziér  
/* i 4 parametri di un rettangolo corrispondono alle coordinate  
dell'angolo superiore sinistro, alla larghezza, all'altezza */  
  
fill(255, 0, 0); // riempimento figure con il rosso  
noStroke(); // non disegnare il contorno  
ellipse(200, 300, 110, 50); // ellisse  
/* i 4 parametri di un'ellisse corrispondono rispettivamente alle  
coordinate del centro, alla larghezza, all'altezza */
```

Il programma è quasi autoesplicativo. Gli unici commenti riguardano i comandi *stroke()* e *fill()*, che stabiliscono il colore di disegno dei contorni e rispettivamente dell'area interna di una figura. I tre numeri interi che costituiscono gli argomenti di queste istruzioni rappresentano un colore in formato RGB (Red, Green, Blue) . Al contrario, i metodi *noStroke()* e *noFill()* inibiscono il disegno del contorno o dell'interno di una figura, fino all'esecuzione di un nuovo comando *stroke()* o *fill()*. Il comando *bezier()* disegna una curva di Beziér, definita da quattro punti: i punti estremi e due punti di controllo che ne determinano la curvatura.

Per comprendere meglio le curve di Beziér, segui [questo link](#).

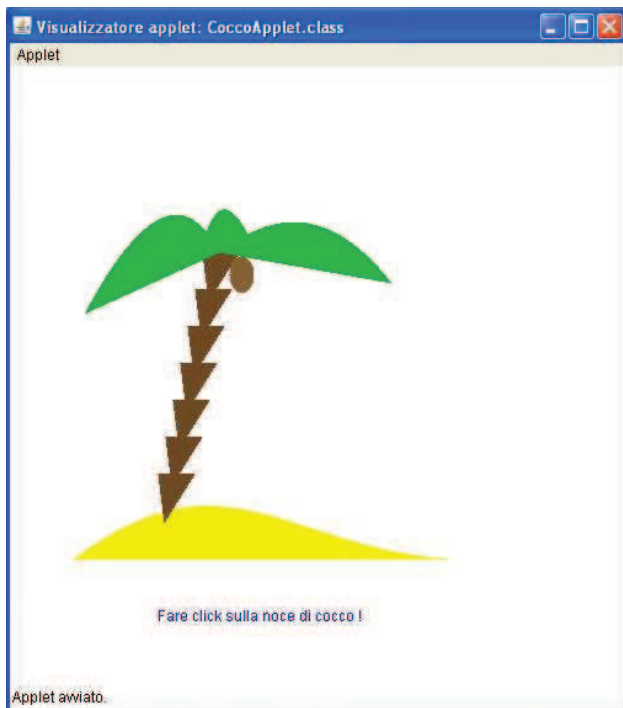
Per determinare il codice RGB di un colore o viceversa per determinare il colore di una tripla RGB di numeri interi, si può usare nel menu "Tools" di Processing l'opzione "Color selector".

Il risultato dell'esecuzione del programma è il seguente



Esercizi

- 1) Scrivere un programma che produce il seguente disegno



2) Scrivere un programma bersaglio che riproduce il disegno seguente:



Suggerimento: usare un ciclo iterativo (for o while). In fondo a questa pagina trovate il riferimento "[disegno bersaglio](#)" a una possibile soluzione. PROVATE A SVOLGERE L'ESERCIZIO SENZA GUARDARE LA SOLUZIONE!

Note sull'esecuzione dei programmi

Come abbiamo visto, Processing dà la possibilità di scrivere ed eseguire programmi in modo immediato. Il codice dei programmi scritti è memorizzato in un file con estensione ".pde" (sigla di Processing Development Environment).

Al momento della sua esecuzione, tale file viene tradotto in un codice detto **bytecode**, contenuto insieme ad altri file all'interno di un file con estensione ".jar" (sigla di Java ARchive).

Il bytecode all'interno del file jar viene eseguito dall'interprete Java, detto anche JRE (sigla di **Java Run-time Environment**) o JVM (sigla di **Java Virtual Machine**).

Le fasi di traduzione (detta comunemente **compilazione**) del codice sorgente (il file pde) e di esecuzione del bytecode avvengono in modo trasparente per l'utente. Tutti i file prodotti mantengono lo stesso nome, cambiando solo l'estensione.

In realtà, il bytecode prodotto da Processing è un programma di tipo speciale, detto **applet**,

che può essere inserito all'interno di una pagina web con un'apposito marcatore <applet>. In questo modo, il programma una volta compilato è inserito all'interno di una pagina web e, al momento della visualizzazione della pagina, l'applet è eseguito. L'unica cosa necessaria per la visualizzazione degli applet è che sul calcolatore sia installato l'interprete Java, (JRE o JVM).

Attenzione, Processing crea l'applet solo se si esegue il comando "export".

In tal caso, esso genera il bytecode dell'applet ed una semplice pagina web di nome index.html contenente un riferimento all'applet mediante il marcatore omonimo.

Di seguito sono elencati i collegamenti alle pagine web contenenti gli esempi visti finora, più alcuni esempi supplementari.

Processing può anche creare un'applicazione se si esegue il comando "export application".

In tal caso, esso genera oltre al bytecode dell'applet un file eseguibile che avvia a sua volta l'esecuzione dell'applet. L'applicazione può essere generata per i sistemi Windows, Mac e Linux.

Di seguito sono elencati i collegamenti alle pagine web contenenti gli esempi visti finora, più alcuni esempi supplementari.

Collegamenti ai programmi discussi

- [esempio sequenza](#)
- [altro esempio sequenza](#)
- [esempio if](#)
- [esempio if-else](#)
- [esempio while](#)
- [esempio for](#)
- [esempio for n. 2](#)
- [esempio di grafica](#)
- [disegno bersaglio](#)

- [numeri triangolari](#)