Matlab: Funzioni

Informatica B

Francesco Trovò

francesco1.trovo@polimi.it

19 Novembre 2021

Perché Usare le Funzioni

A Cosa Servono le Funzioni?

```
x = input('inserisci x: ');
fx = 1
for ii = 1 : x
  fx = fx * ii;
end
if (fx > 220)
  y = input('inserisci y: ');
  fy = 1
  for ii = 1 : y
    fy = fy * ii;
  end
end
```

Entrambi i frammenti di codice eseguono il calcolo del fattoriale

A cosa Servono le Funzioni?

Riusabilità

- Scrivo una sola volta codice utilizzato spesso
- Modifiche e correzioni sono gestibili facilmente
- Lo stesso codice viene facilmente richiamato in diversi programmi

Leggibilità

- Incapsulo porzioni di codice complesso, il programmatore non deve entrare nei dettagli
- Aumento il livello di astrazione dei miei programmi

Flessibilià

- Posso aggiungere funzionalità non presenti nelle funzioni di libreria

Usiamo uno Script-file?

Soluzione semplice: uno script file può essere usato per incapsulare porzioni di codice riusabili in futuro

```
x = input('inserisci x: ');
fx=1
for ii=1:x
  fx = fx*ii
end
if (fx > 220)
  y = input('inserisci y: ');
  fy=1
  for ii=1:y
    fy = fy*ii
  end
end
```

fattoriale.m

```
f = 1;
for ii = 1:n
  f = f*ii
end
```

Problemi:

- Come fornisco l'input allo script?
- Dove recupero l'output?

Soluzione con gli Script-files

Gli script utilizzano le variabili del workspace:

```
x = input('inserisci x: ');
n=x;
fattoriale;
fx=f;
if (fx > 220)
   y = input('inserisci y: ');
   n=y;
   fattoriale;
   fy=f;
end
```

Prepara l'input in n

Chiama lo script

Salva il risultato in f

Prepara l'input in n

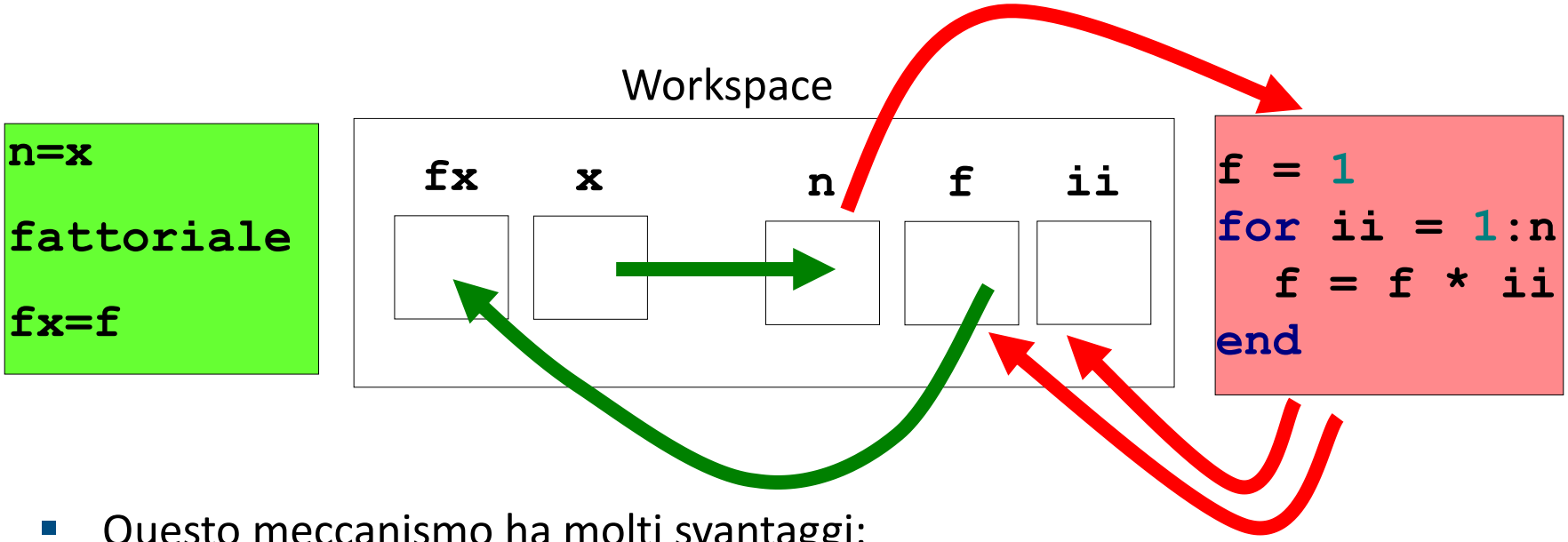
Chiama lo script

Salva il risultato in f

fattoriale.m

```
f = 1
for ii = 1:n
   f = f*ii
end
```

Limiti degli Script-files



- Questo meccanismo ha molti svantaggi:
  - poco leggibile
  - richiede molte istruzioni
  - poco sicuro
- Tutte le variabili sono nello stesso workspace: fattoriale.m può modificare tutte le variabili del workspace, ad esempio se **`ii`** fosse usata nel main questa sarebbe sovrascritta

Le Funzioni: Definizione

header

```
function f = fattoriale(n)
  f = 1
  for ii = 1:n
    f = f * ii
  end
```

body

`n` è **l'argomento** della funzione (serve a fornire l'input)

`f` è il **valore restituito** dalla funzione (serve a fornire l'output)

- La testata (header) inizia con la parola chiave function e definisce:
  - nome della funzione
  - argomenti (input)
  - valore restituito (output)
- Il corpo (body) definisce le istruzioni da eseguire quando la funzione viene chiamata:
  - utilizza gli argomenti e assegna il valore di ritorno

Le Funzioni: Ulteriori Dettagli

Una funzione può avere più argomenti separati da virgola:

```
function f(x, y)
```

Nel caso sia necessario restituire più valori, definiamo l'header affiancando più variabili in output usando la stessa notazione degli array:

```
function  [v1, v2 , ...] = f(x, y)
```

Es.

```
function [s, p] = sumProd(a, b)
    s = a + b;
    p = a * b;
```

Sintassi per la Definizione di una Funzione

La sintassi per definire l'header di funzione è:

```
function [out1,...,outM] = nomeFunzione(in1,...,inN)
```

- **Gli argomenti (parametri in ingresso)** `in1,...,inN` vanno elencate tra parentesi tonde e seguono il nome della funzione
- **I valori restituiti (parametri in uscita)** `out1,...,outN` vanno elencate tra parentesi quadre e seguono la keyword `function`

NB: la notazione `[out1,...,outM]` per le variabili in uscita di una funzione è la stessa dell'operatore CAT orizzontale, ma i valori restituiti possono avere dimensioni e tipi non consistenti

NB: se la funzione non ha parametri in ingresso/uscita le parentesi tonde rimangono vuote/quadre si possono omettere

Invocazione

- Una funzione può essere invocata in un programma attraverso il suo nome, seguito dagli argomenti fra parentesi rotonde
- La funzione viene quindi eseguita e il suo valore restituito viene calcolato

Es.

```
x = input('inserire x:');
fx = fattoriale(x);
if (fx > 220)
   y = input('inserisci y: ');
   fy = fattoriale(y);
end
```

```
function f = fattoriale(n)
  f = 1
  for ii = 1:n
    f = f * ii
  end
```

I Parametri: Formali vs. Attuali

Definizioni:

- I **parametri formali** sono le variabili usate come **argomenti** e **valori** di **ritorno** **nella definizione** della funzione
- I **parametri attuali** sono i valori (o le variabili) usati come **argomenti** e come **valori restituiti** **nell'invocazione** della funzione

```
function f = fattoriale(n)
   f = 1;
   for ii = 1:n
      f = f*ii;
   end
>> fat5 = fattoriale(5) %Invocazione
fat5 =
    120
```

f ed **n** sono parametri formali

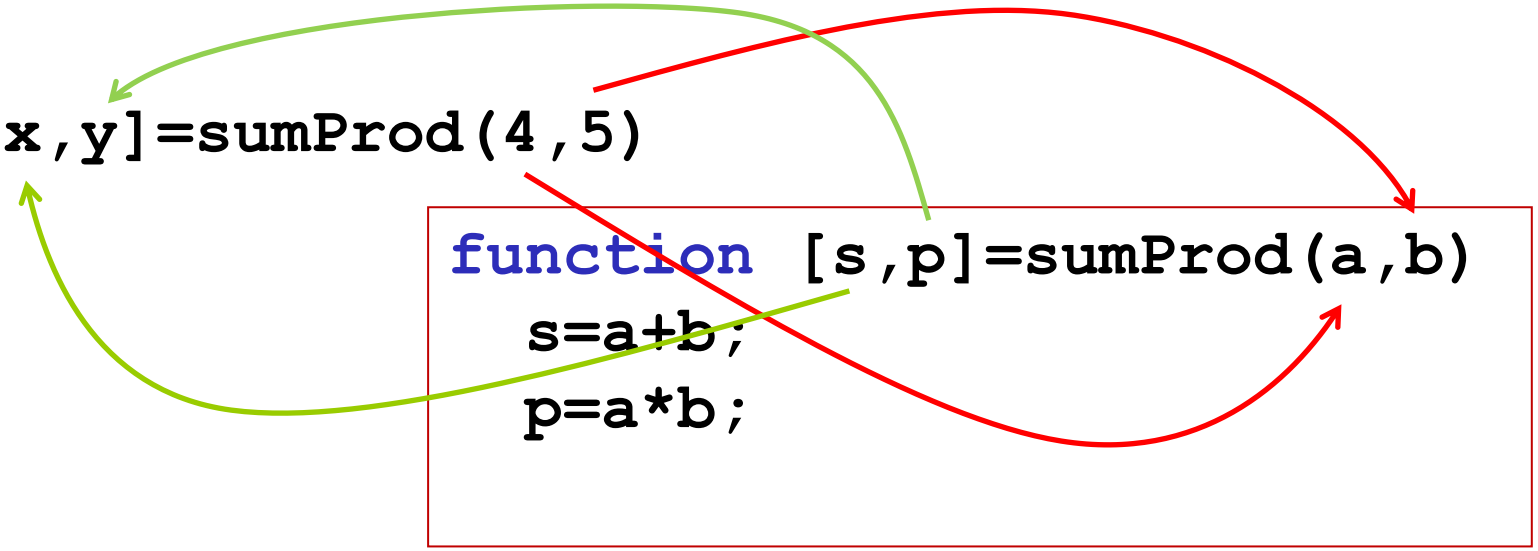
fat5 e **5** sono parametri attuali

L'ordine dei Parametri

- **Qualsiasi tipo di parametro** è ammesso (scalari, vettori, matrici, etc.)
- I **parametri attuali** vengono **associati a quelli formali** in **base alla posizione**: il primo parametro attuale viene associato al primo formale, il secondo parametro attuale al secondo parametro formale, etc.

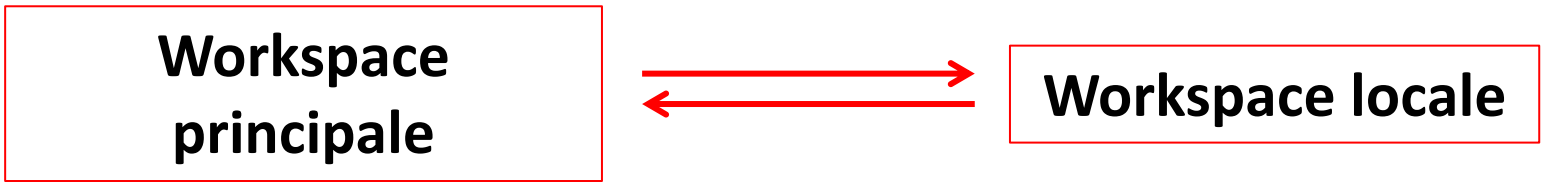
Es.

```
>> [x,y]=sumProd(4,5)
```



Esecuzione di una Funzione

- Quando una funzione viene eseguita, viene creato un workspace locale in cui vengono memorizzate tutte le variabili usate nella funzioni inclusi i parametri formali
  - All'interno delle funzioni non si può accedere al workspace principale (nessun conflitto di nomi)
  - Al termine dell'esecuzione della funzione, il workspace locale viene distrutto!



Le comunicazioni tra i workspace avvengono solamente mediante **copia dei valori** dei parametri in ingresso ed in uscita

Esecuzione di una Funzione Passo Passo

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato un workspace locale** per la funzione
3. I **valori** dei **parametri attuali** di ingresso vengono **copiati** nei **parametri formali** all'interno del **workspace locale**, che ora contiene solamente i parametri formali con assegnati i valori dei parametri attuali
4. Viene **eseguito il corpo** della **funzione**
5. Vengono **copiati i valori restituiti dai parametri formali nel workspace locale al workspace principale** nei corrispondenti parametri attuali
6. Il workspace "locale" viene **distrutto**

Esecuzione di una Funzione: Esempio

```
%Main script
(1) >> x = 3;
(2) >> w = 2;
(3) >> r = funz(4);
```

```
function y = funz(x)
  y = 2 * x;  %(1')
  x = 0;      %(2')
  z = 4;      %(3')
```

W principale dopo (2)

```
x = 3
w = 2
```

W locale dopo (1')

```
x = 4
y = 8
```

W principale dopo (3)

```
x = 3
w = 2
r = 8
```

W locale dopo (3')

```
x = 0
y = 8
z = 4
```

~~W locale dopo (3)~~

Esecuzione di una Funzione: Esempio

```
%Main script
(1) >> x = 3;
(2) >> w = 2;
(3) >> r = funz(4);
```

```
function y = funz(x)
  y = 2 * x;  %(1')
  x = 0;      %(2')
  z = 4;      %(3')
  x = w - 1;  %(4')
```

W principale dopo (2)

```
x = 3
w = 2
```

W locale dopo (1')

```
x = 4
y = 8
```

W locale prima (4')

```
x = 0
y = 8
z = 4
w = ? %errore
```

W principale dopo (3)

```
x = 3
w = 2
```

W locale dopo (3')

```
x = 0
y = 8
z = 4
```

~~W locale dopo (3)~~

Numero dei Parametri

- Il **numero di parametri attuali** all'invocazione della funzione deve essere identico al numero di **parametri formali in ingresso**
- Il **vincolo vale per i parametri in ingresso**, anche se è possibile trattare i parametri formali nella funzione per gestire questi casi

- Il **vincolo non vale per i parametri in uscita**: verranno assegnati solamente i parametri attuali specificati
  - Ad esempio `s = sommaProd(5,2)` il valore della somma viene assegnato a `s` ma non il valore del prodotto (anche se la funzione lo calcola)

Esercizio

Scrivere una funzione che prende in ingresso tre numeri e resituisce il massimo ed il minimo

```
function [mini, maxi] = minmax(a, b, c)
maxi = a;
if maxi < b
 maxi = b;
end
if maxi < c
 maxi = c;
end
```

```
mini = a;
if mini > b
 mini = b;
end
if mini > c
 mini = c;
end
```

Esempio

Scrivere una funzione che prende in ingresso tre numeri e resituisce il massimo ed il minimo

```
function [mini, maxi] = minmax(a, b, c)
maxi = max([a, b, c]);
mini = min([a, b, c]);
```

Note sui Parametri in Uscita

I parametri formali dei valori restituiti devono essere **sempre definiti** (eventualmente possono essere vuoti)

Es.

```
function [positivi, media] = mediaPositivi(vett)
somma = 0; cnt = 0;
positivi = [];
for ii = 1 : length(vett)
  if vett(ii) > 0
    positivi = [positivi, vett(ii)];
    somma = somma + vett(ii);
    cnt = cnt + 1;
  end
end
if cnt > 0
  media = somma / cnt;
end
```

```
>> [a,b] = mediaPositivi(-[1 : 10])
Error in mediaPositivi (line 2)
positivi = vett(vett >0);

Output argument "media" (and maybe
others) not assigned during call to
mediaPositivi
```

Soluzione Esercizio Precedente

```
function [positivi, media] = mediaPositivi(vett)
somma = 0; cnt = 0;
positivi = [];
for ii = 1 : length(vett)
    if vett(ii) > 0
        positivi = [positivi, vett(ii)];
        somma = somma + vett(ii);
        cnt = cnt + 1;
    end
end
if cnt > 0
    media = somma / cnt;
else
    media =  [];
end
```

Restituisco comunque la variabile media anche se vuota

Note sull'Output

```
function [s, p] = sumProd(a, b)
  s = a + b;
  p = a * b;
```

- È possibile invocare la funzione senza specificare due parametri in uscita:
  - **`x = sumProd(4,5)`** assegna ad **`x`** solamente il primo output
  - **`[~, y] = sumProd(4,5)`** assegna ad **`y`** solo il secondo output

File Funzione

- Come nel caso degli script le funzioni sono scritte in file di testo sorgenti:
  - devono avere estensione .m
  - devono avere lo stesso nome della funzione
  - la prima riga del file deve contenere l'header della funzione e di fatto iniziare con la parola chiave **function**

NB: non ridefinire funzioni esistenti (usare `exist('nomeFunzione')` per controllare se una funzione esiste

- Se commentate, le prime righe della funzione rappresentano l'help e vengono visualizzate quando si scrive:

```
>> help nomeFunzione
```

Esercizio

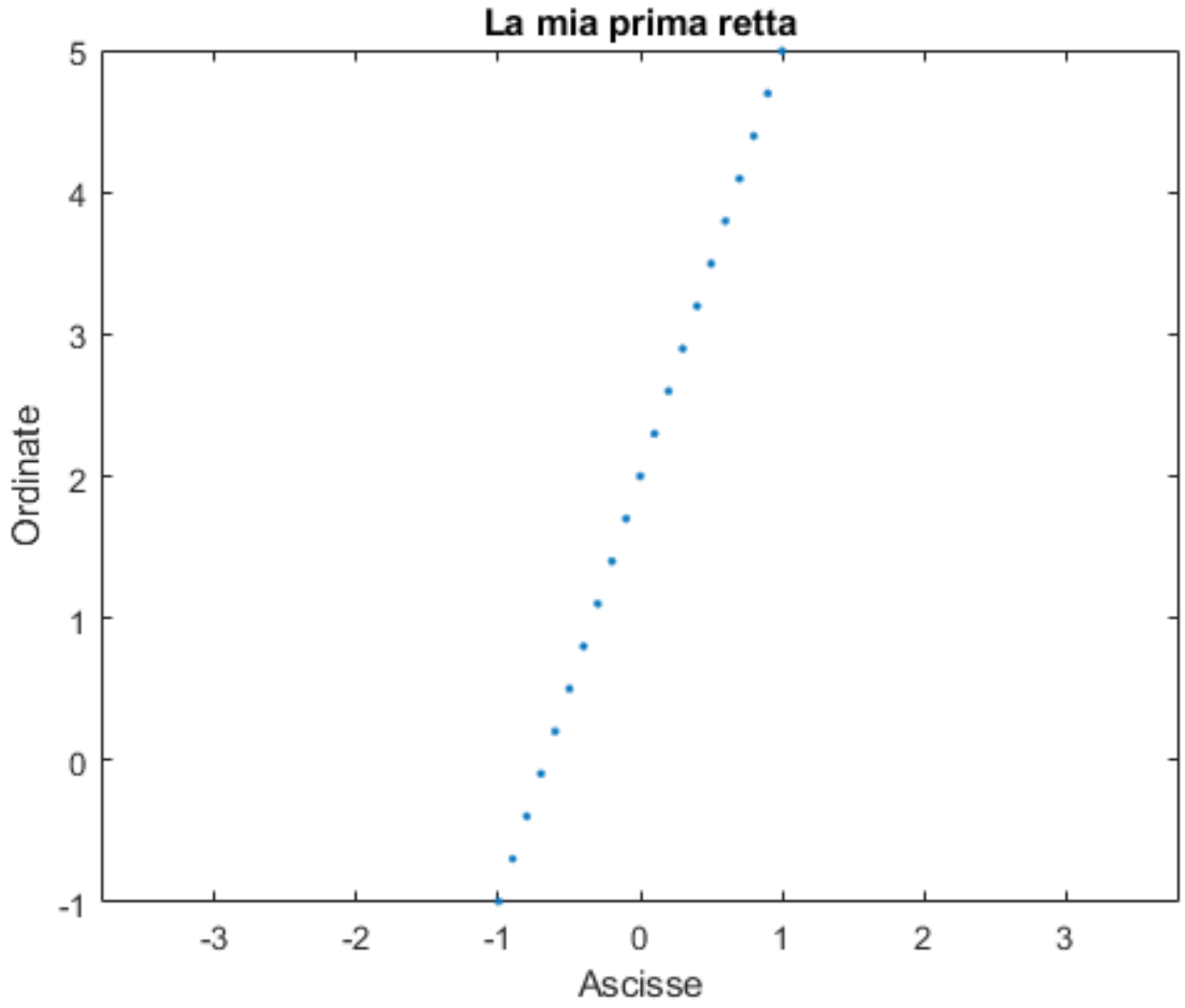
Scrivere una funzione che prende in ingresso due coefficienti **m** e **q** ed un vettore di punti **xx** e restituisce il vettore **yy** dei punti che stanno sulla retta $y = mx + q$ in corrispondenza a **xx**

```
function yy = retta(m, q, xx)
% function yy = retta(m, q, xx)
% INPUT
% m, q: coefficienti
% xx: vettore di punti
% OUTPUT
% yy: ordinate dei punti xx della retta y =
mx + q

yy = m * xx + q;
```

Esempi di Script per Invocare la Funzione

```
x = -1 : 0.1 : 1;
% invoco la funzione per plottare y = 3x + 2
y = retta(3, 2, x);

figure();
plot(x, y, '.');
axis equal;
title('La mia prima retta');
xlabel('Ascisse');
ylabel('Ordinate');
```

Risultato dello Script



Esercizio

Scrivere una funzione che calcola la sequenza di Fibonacci della lunghezza richiesta

La successione di Fibonacci è definita così:

- $F(0) = 0$
- $F(1) = 1$
- $F(n) = F(n-1) + F(n-2), n > 1$

```
function F = fibonacci(n)
% function F = fibonacci(n)
%
% restituisce un vettore (F) contenente
%  i primi n numeri di fibonacci
F = [0 , 1];
for indx  = [3 : 1 : n]
    F(indx) = F(indx - 1) + F(indx - 2);
end
```

Esempio

Scrivere una funzione *contoAllaRovescia* che prende in ingresso un intero (che esprime i secondi) ed esegue il conto alla rovescia

Al termine viene emesso un suono e mandato un messaggio a schermo

```
function contoAllaRovescia(n)
disp(['... ', num2str(n)])
for ii = [n - 1: -1 : 0]
    pause(1)
    disp(['... ', num2str(ii)])
end
disp('Boom!')
load handel;
sound(y, Fs);
```

Se la funzione non restituisce nulla si può omettere `[] =`

Esercizio

Scrivere un programma che chiede all'utente di inserire un numero positivo (controllare che sia positivo)

- Verificare se il numero è perfetto
- In caso contrario dire se è abbondante o difettivo

Richiede quindi un altro numero

- controlla se i due numeri sono amici

Un numero è:

- perfetto se corrisponde alla somma dei suoi divisori, escluso se stesso
- abbondante se è maggiore della somma dei suoi divisori
- difettivo se è inferiore alla somma dei suoi divisori
- a e b sono amici se la somma dei divisori di a è uguale a b e viceversa

Implemento Diverse funzioni che Richiamo

```
function n = inserisciInteroPositivo()
% function n = inserisciInteroPositivo()
% richiede all'utente di inserire un intero positivo
% e lo restituisce

function somma = calcolaSommaDivisori(n)
%function somma = calcolaSommaDivisori(n)
% calcola la somma di tutti i divisori di n escluso n

function [res, abb] = controllaSePerfetto(n)
% function [res, abb] = controllaSePerfetto(n)
% res = true se n è perfetto (uguale alla somma dei suoi
divisori escluso se stesso)
%  se res = false e abb = true/false se è abbondante o
difettivo

function res = controllaSeAmici(a,b)
% function res = controllaSeAmici(a,b)
% res = 1 se a è amico di b, 0 altrimenti
```

Funzione (1)

```
function n = inserisciInteroPositivo()
% function n = inserisciInteroPositivo()
% richiede all'utente di inserire un intero positivo
% e lo restituisce

isPositivo = 0

while(isPositivo == 0)
    n = input('Inserire intero positivo: ')
    isPositivo = (n > 0 && n == round(n));
%     if (n > 0 && n == round(n))
%         isPositivo = 1;
%     else
%         isPositivo = 0;
%     end
end
```

Funzione (2)

```
function [res, abb] = controllaSePerfetto(n)
% function [res, abb] = controllaSePerfetto(n)
%  res = true se n è perfetto
%  se res = false, abb = true/false se è
abbondante/difettivo

s = calcolaSommaDivisori(n); % assegno ad s ed evito 2
chiamate
abb = []; % è necessario per quando res ==false
if (n == s)
    res = true;
else
    res = false;
    if n > s
        abb = true;
    else
        abb = false;
    end
end
```

Funzione (3)

```
function res = controllaSeAmici(a,b)
% function res = controllaSeAmici(a,b)
% res = 1 se a è amico di b, 0 altrimenti

if b == calcolaSommaDivisori(a) && a ==
calcolaSommaDivisori(b)
    res = true;
else
    res = false;
end
```

Funzione (4)

```
function somma = calcolaSommaDivisori(n)
%function somma = calcolaSommaDivisori(n)
% calcola la somma di tutti i divisori di n
escluso n

somma = 0;
for ii = 1 : n / 2 % inutile andare oltre n/2
    if (mod(n, ii) == 0)
        somma = somma + ii;
    end
end
```

Main Script

```
n = inserisciInteroPositivo();
[perf, abbond] = controllaSePerfetto(n);
if(perf == true)
    disp([num2str(n), ' è perfetto']);
else
    disp([num2str(n), ' NON è perfetto']);
    if(abbond == true)
        disp([num2str(n), ' è abbondante']);
    else
        disp([num2str(n), ' è difettivo']);
    end
    m = inserisciInteroPositivo();
    amici = controllaSeAmici(n,m);
    if(amici)
        disp([num2str(n), ' e ', num2str(m), ' sono amici'])
    else
        disp([num2str(n), ' e ', num2str(m), ' NON sono amici'])
    end
end
```