Funzioni

Informatica (ICA) AA 2020 / 2021

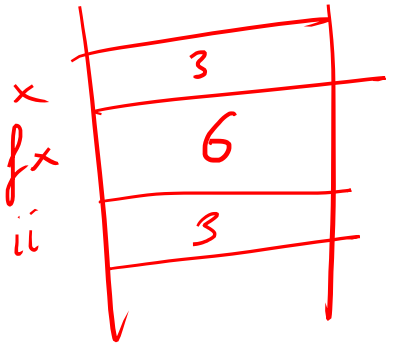
Giacomo Boracchi

30 Ottobre 2020

giacomo.boracchi@polimi.it

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
```



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

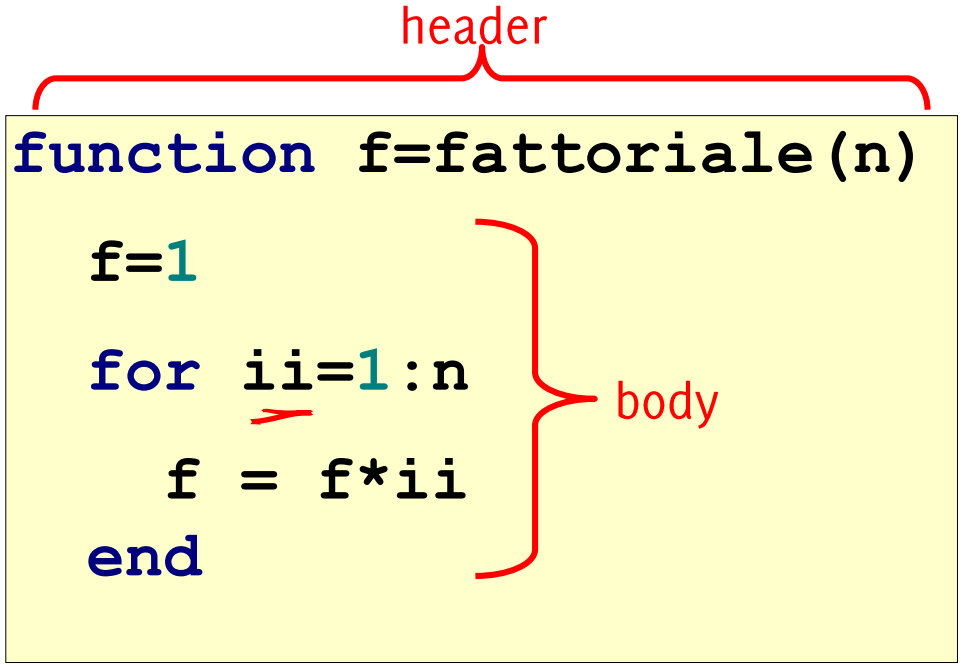
- Incapsulo porzioni di codice complesso, il programmatore non deve entrare nei dettagli
- Aumento il livello di astrazione dei miei programmi

Flessibilià

- Posso aggiungere funzionalità non presenti nelle funzioni di libreria

Le Funzioni

Le funzioni



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

f è il **valore di ritorno** della funzione (serve a fornire l'output)

- L' header inizia con la parola chiave **function** e definisce:
  - nome della funzione
  - argomenti (input)
  - valore di ritorno (output)
- Il corpo definisce le istruzioni da eseguire quando la funzione viene chiamata
  - Utilizza gli argomenti e assegna il valore di ritorno

Le funzioni (2)

Una funzione può avere più argomenti separati da virgola:

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

Nel caso sia necessario ritornare più valori, definiamo l'header affiancando più variabili in output usando la stessa notazione degli array (attenzione!):

```
function  [v1,v2,…] = f(x,y)
```

Esempio:

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

Definizione dell'header 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 ritornati (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. Però qui ha un altro significato perché `out1, .., outM` possono avere dimensioni e tipi non consistenti!

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 di ritorno viene calcolato.

Esempio

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

Invocazione

Invocazione

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

I Parametri

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** di **ritorno 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

fx e 5 sono parametri attuali

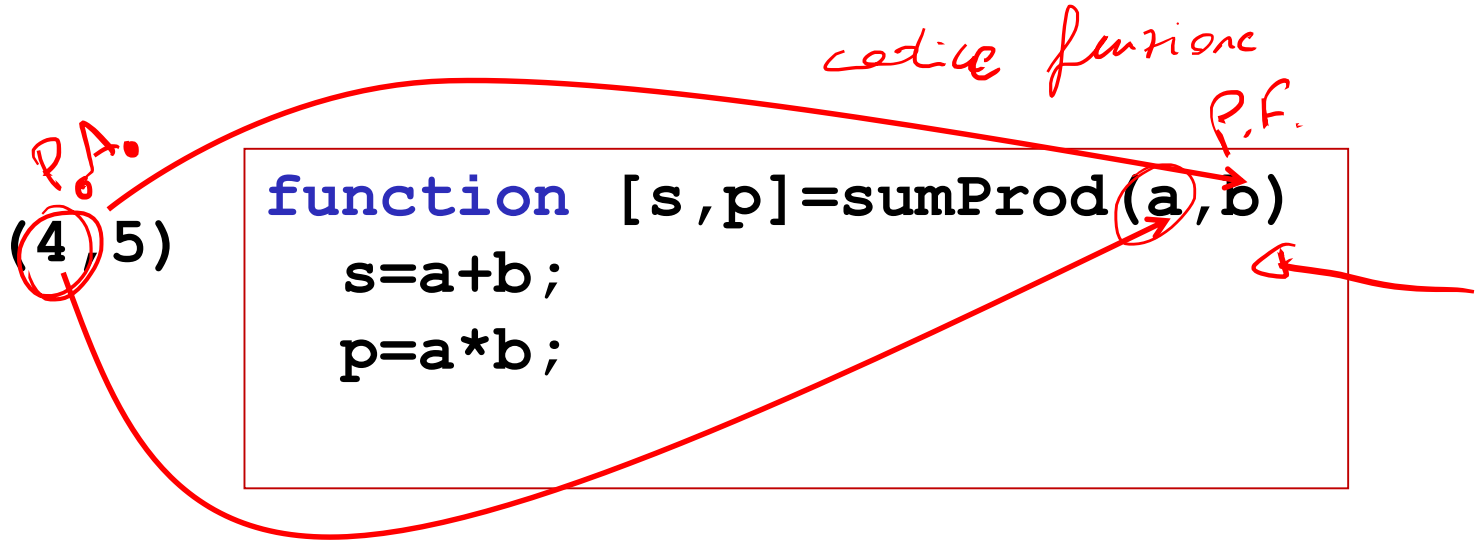
I Parametri (2)

Qualsiasi tipo di parametri è ammesso (scalari, vettori, matrici, strutture, ecc.)

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, ecc.

Esempio

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

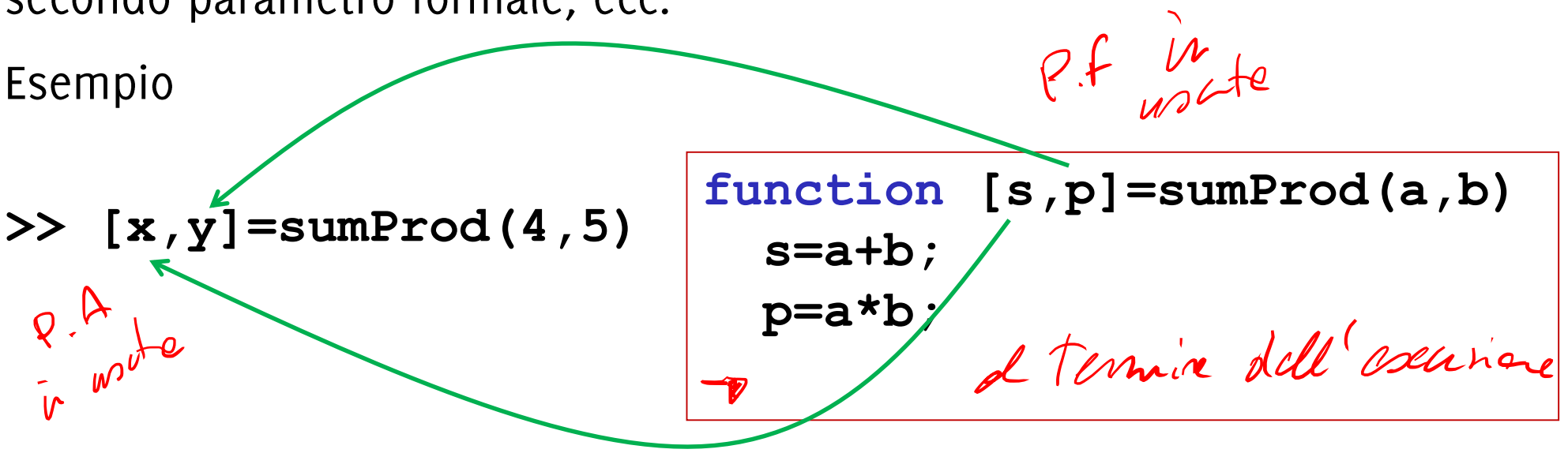


I Parametri (2)

Qualsiasi tipo di parametri è ammesso (scalari, vettori, matrici, strutture, ecc.)

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, ecc.

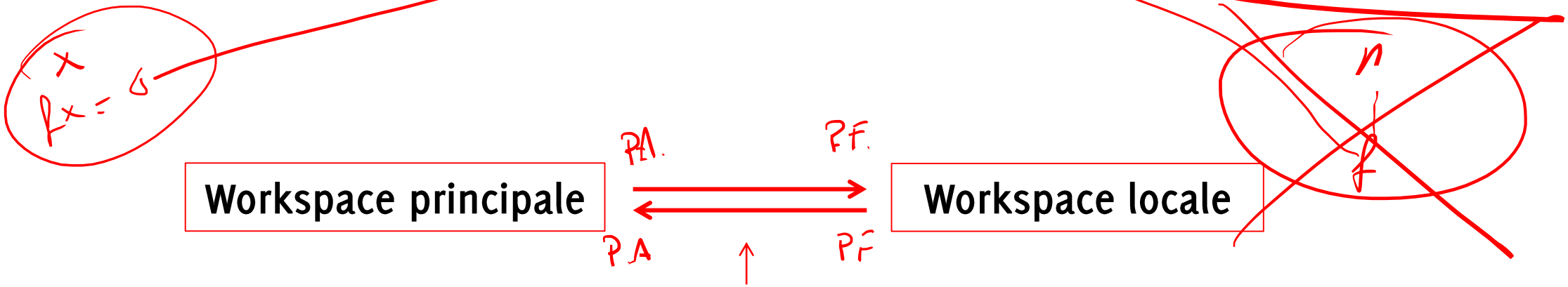
Esempio



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.

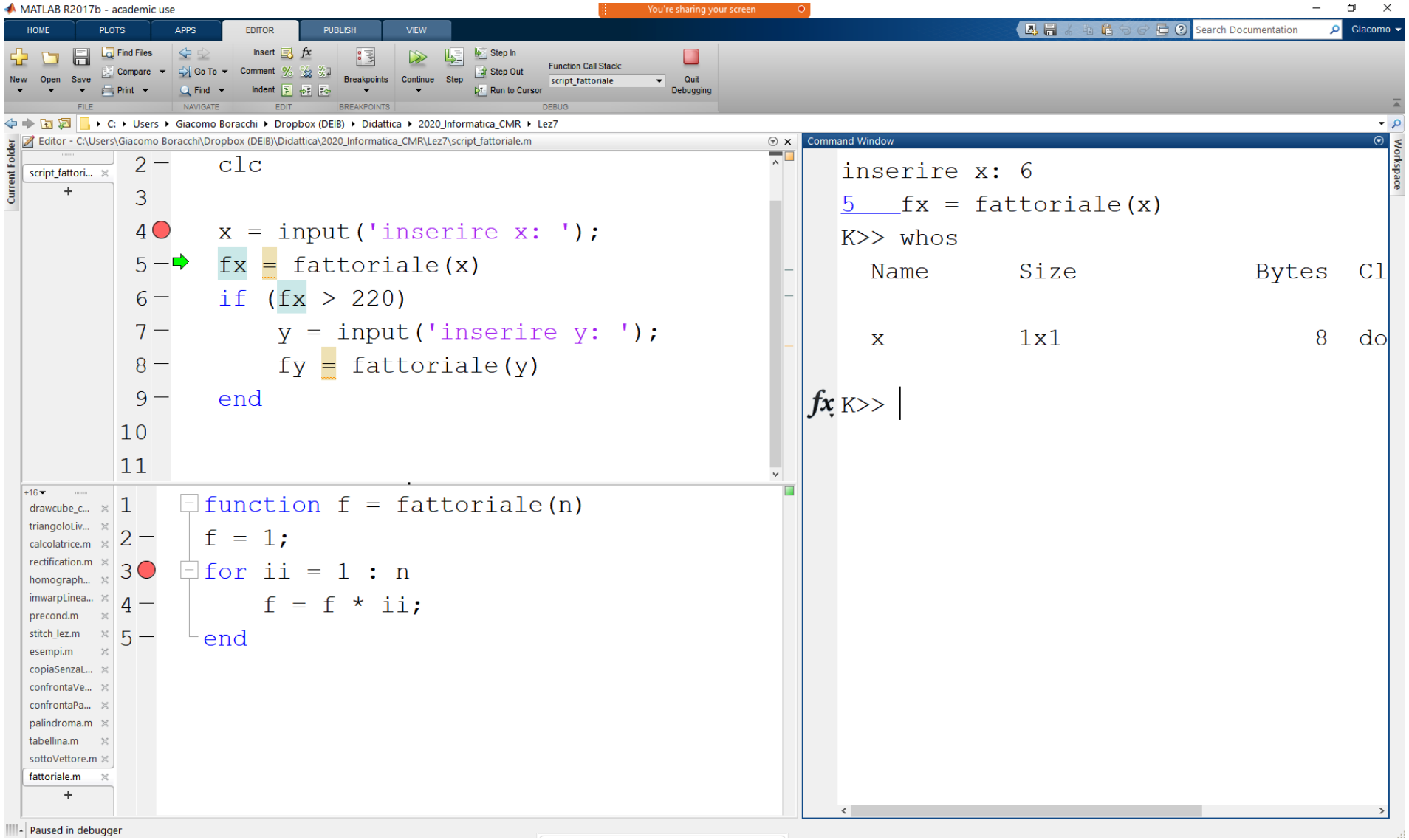
- All’interno delle funzioni non si può accedere al workspace “principale” (nessun conflitto coi nomi delle variabili)
- Al termine dell’esecuzione della funzione, il workspace “locale” viene distrutto!



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

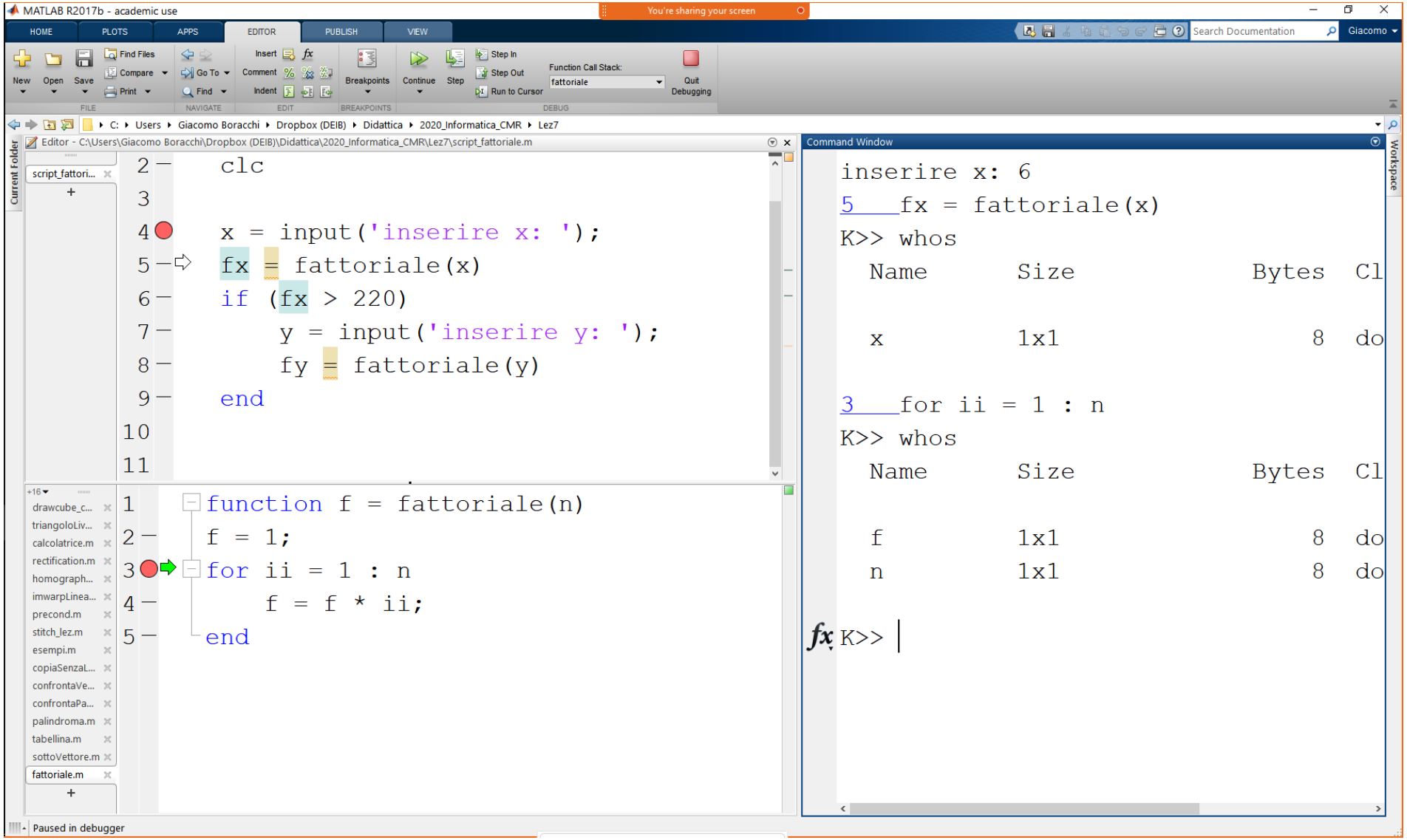
Debugging

Workspace principale prima dell'invocazione della funzione



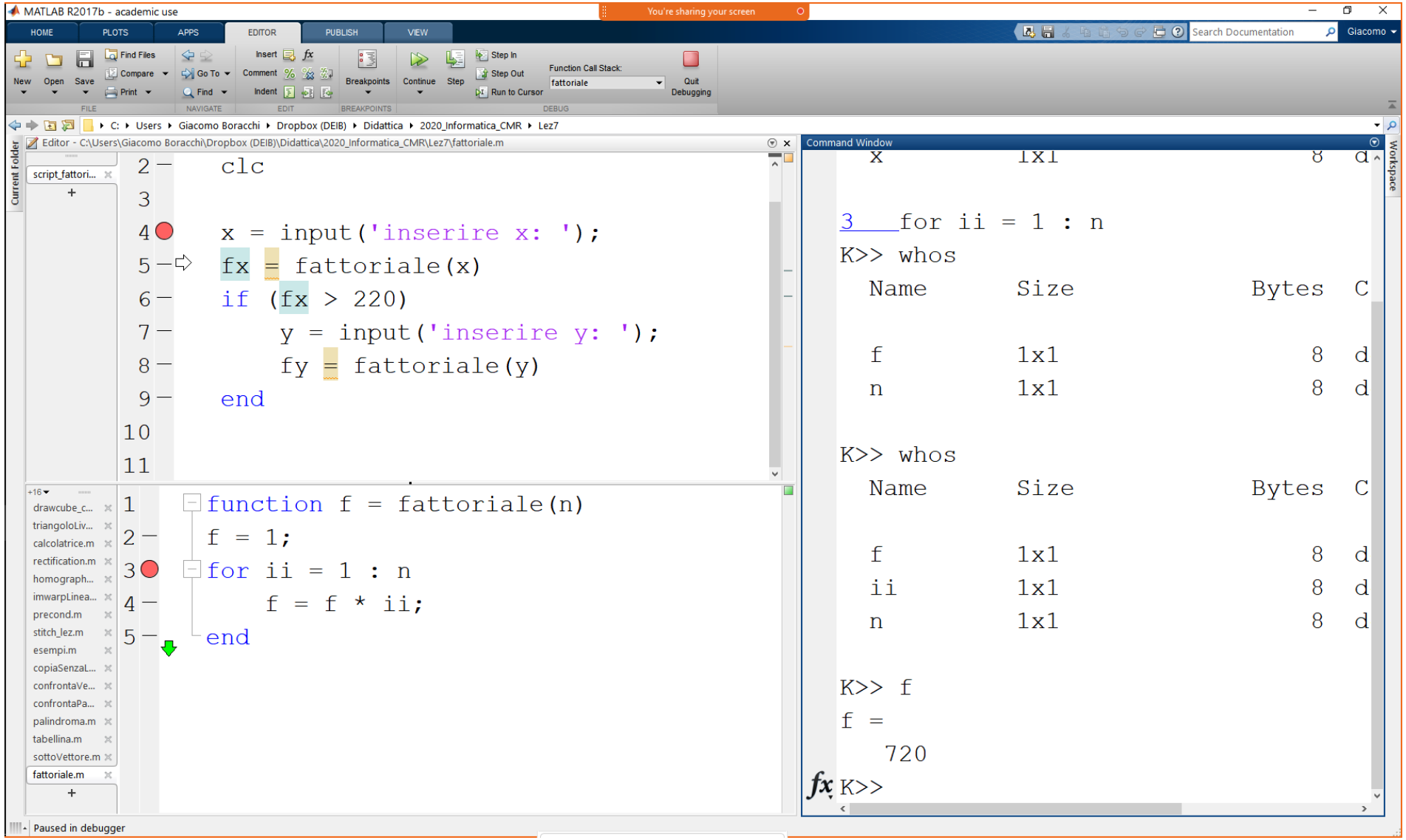
Debugging

Workspace della funzione fattoriale alla prima invocazione. La freccia indica dov'è passato il flusso. I workspace sono separati



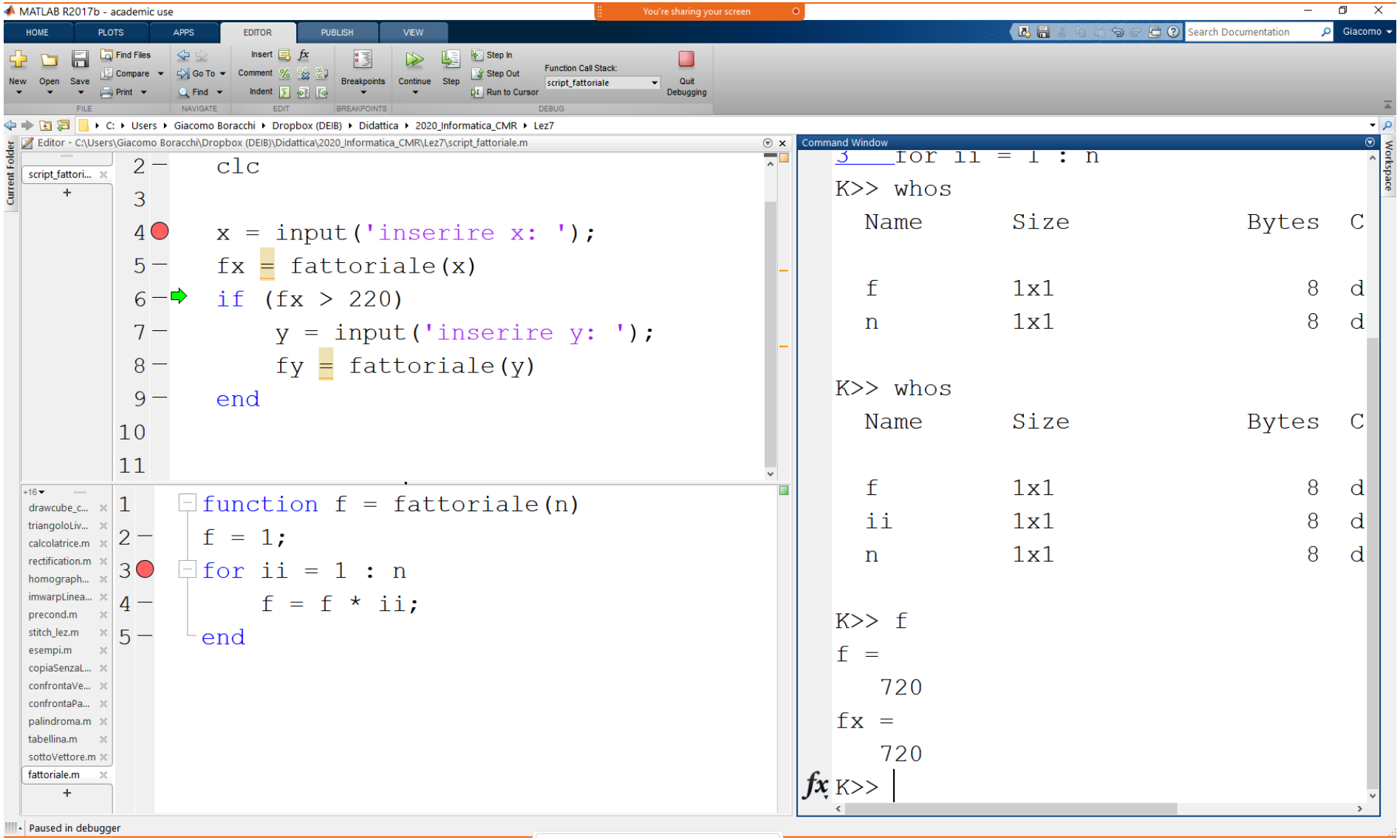
Debugging

Workspace locale prima di restituire al chiamante il valore del parametro formale in uscita



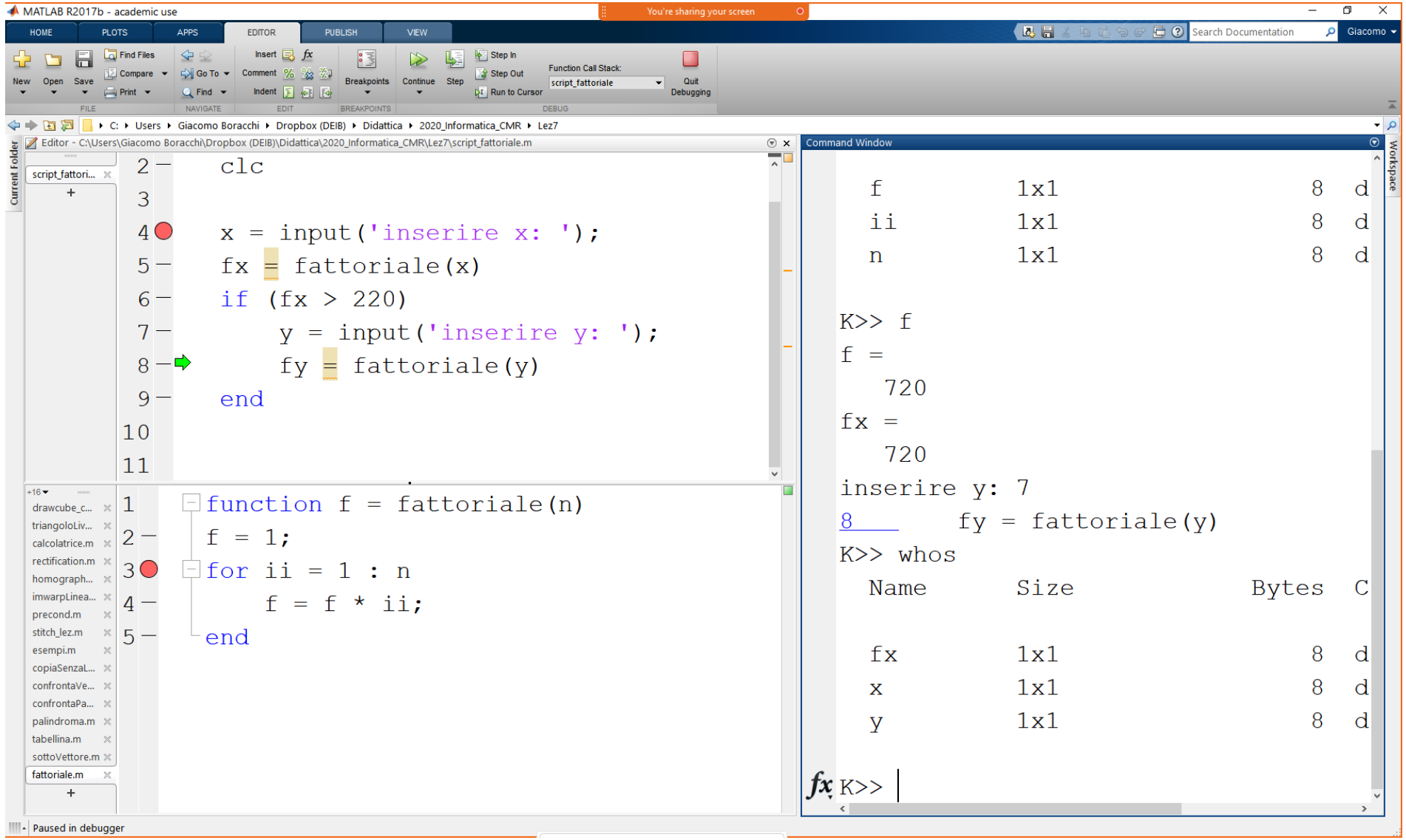
Debugging

Workspace principale dopo l'invocazione della funzione



Debugging

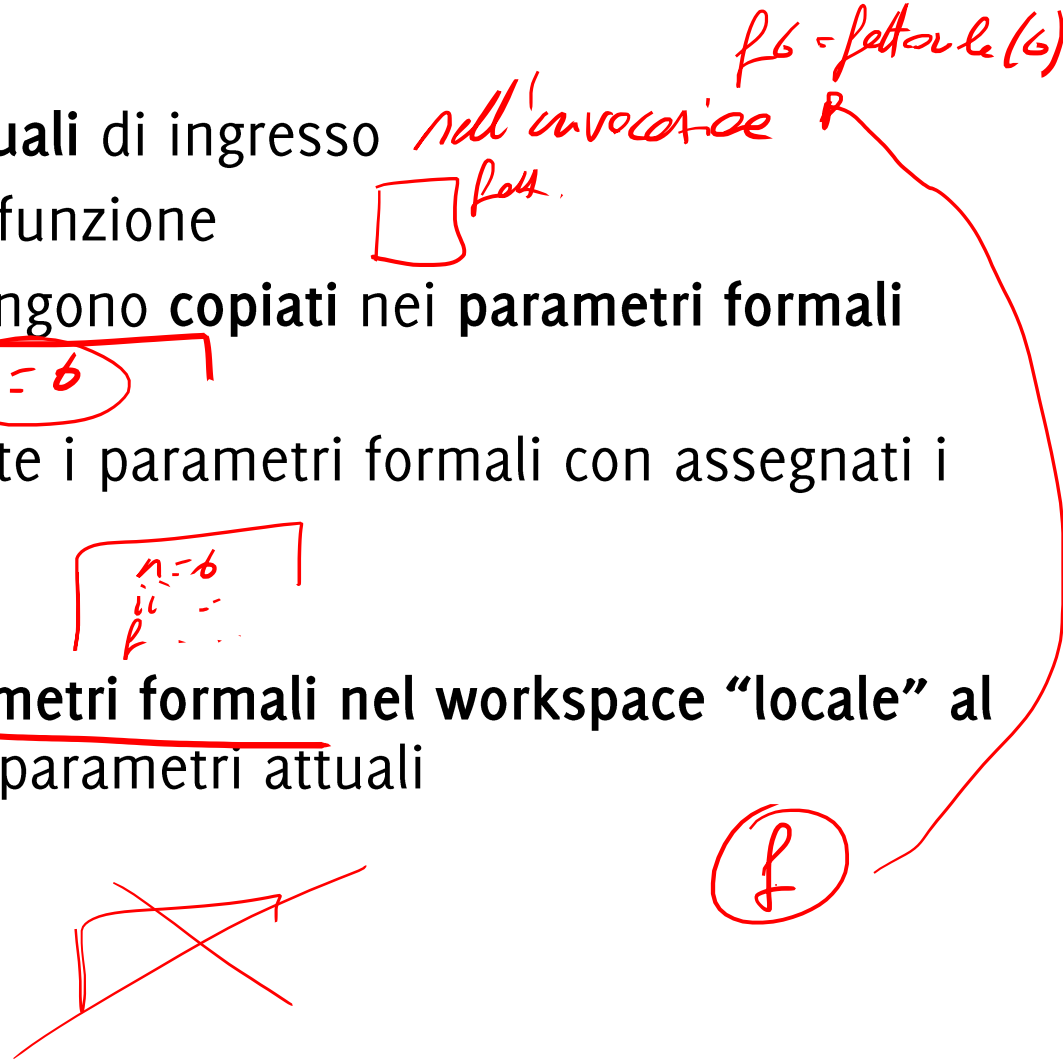
Workspace della funzione fattoriale alla prima della seconda invocazione. Le variabili del workspace locale dalla prima invocazione non compaiono, perché questo è stato distrutto



Riepilogando: Esecuzione di una funzione (2)

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"**
   - Il workspace locale 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 di ritorno dai parametri formali nel workspace "locale" al workspace "principale"** nei corrispondenti parametri attuali
6. Il workspace "locale" viene **distrutto**



Esecuzione di una funzione: esempio

```
(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

```
(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)~~

I Parametri (3)

In linea di massima, **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

I Parametri (3)

In linea di massima, **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)

Esempio

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

Esempio

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

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

Esempi di invocazione

```
>> [minore, maggiore] = minmax(7, 8, 9);
```

```
>> [minore] = minmax(3*x -y, a-1, a);
```

```
>> [~, maggiore] = minmax(s, t, s-t);
```

non è possibile invocare una funzione con con meno parametri in ingresso

```
>> [minore, maggiore] = minmax(s, t);
```

Note sui Parametri in Uscita

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

Questa funzione da errori quando il vettore inserito contiene solamente elementi negativi

```
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
```

Note sui Parametri in Uscita

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

Questa funzione da errori quando il vettore inserito contiene solamente elementi negativi

```
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

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

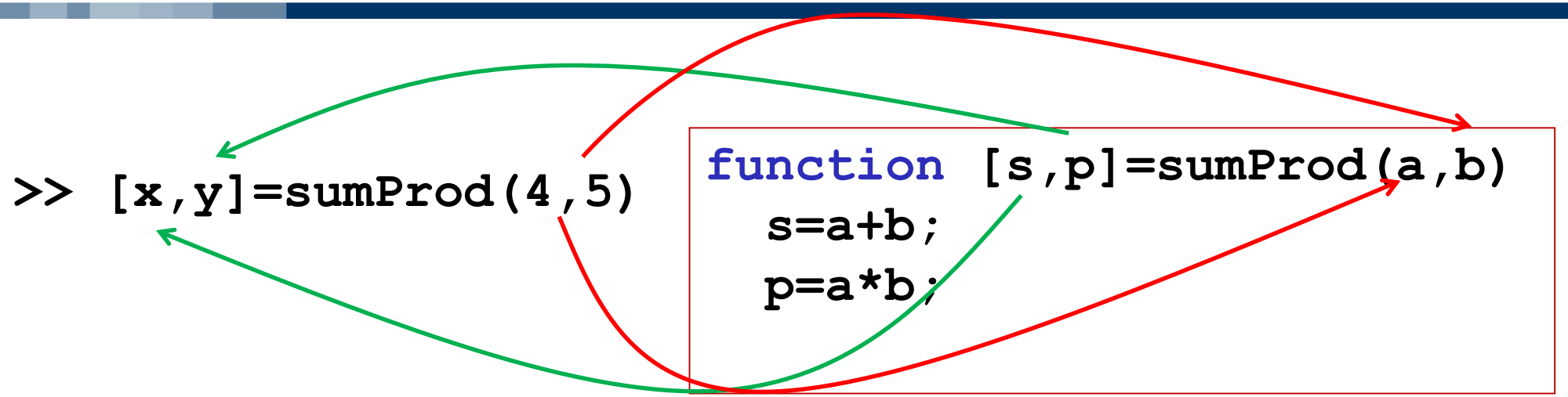
Note sui Parametri in Uscita

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

Questa funzione da errori quando il vettore inserito contiene solamente elementi negativi

```
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
```

Note sull'output



È però possibile invocare la funzione senza specificare due parametri in uscita,

- es **`x = sumProd(4,5)`**. In tal caso solamente il primo output viene assegnato ad **`x`**

L'invocazione **`sumProd(4,5)`** associa alla variabile **`ans`** il primo argomento restituito da **`sumProd`**

Per riceve solo il secondo output uso **`~`** come se fosse una variabile da non considerare **`[~,y] = sumProd(4,5)`**

File funzione

Come nel caso degli script le funzioni possono essere scritti in file di testo sorgenti

- Devono avere estensione .m
- Devono avere lo stesso nome della funzione
- Devono iniziare con la parola chiave **function**

Attenzione a non “ridefinire” funzioni esistenti

- exist(‘nomeFunzione’) → o se la funzione non esiste

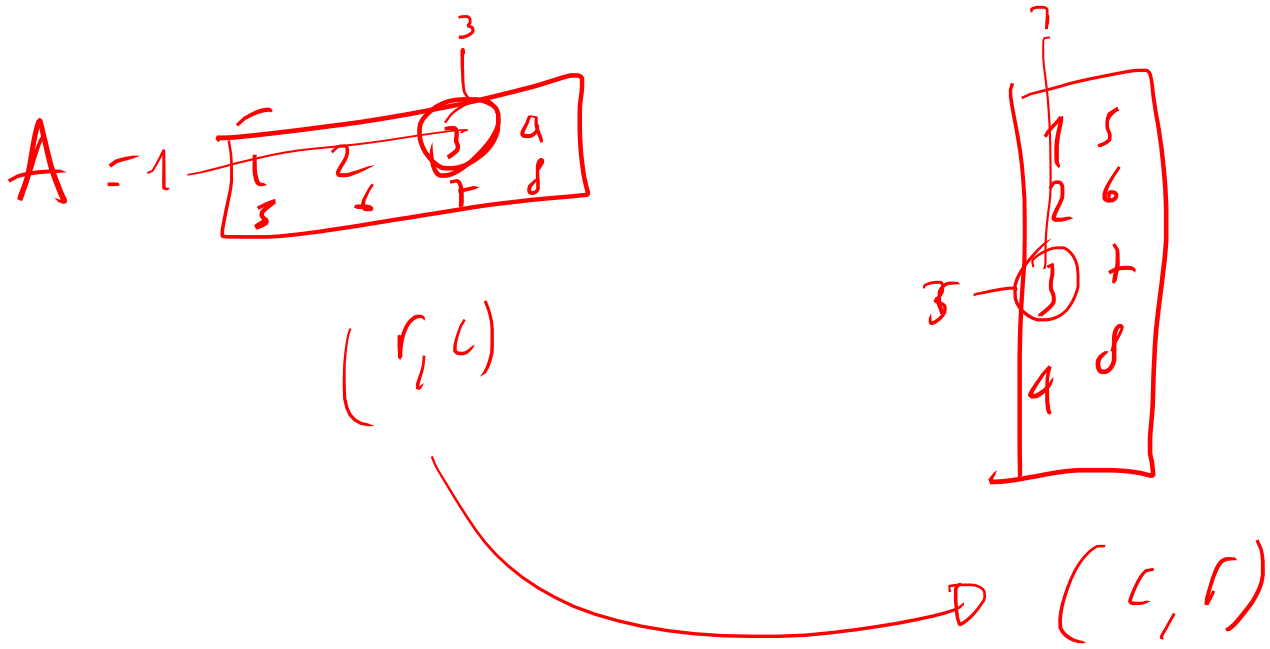
Se commentate, le prime righe della funzione rappresentano l’help e vengono visualizzate quando si scrive: help nomeFunzione

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.

Esercizio

Implementare la funzione trasposizione per le matrici



Esercizio

Implementare la funzione trasposizione per le matrici

```
function [t]=trasposta(m)
    [r,c]=size(m);
    for ii=1:r
        for j=1:c
            t(j,ii)=m(ii,j);
        end;
    end
```

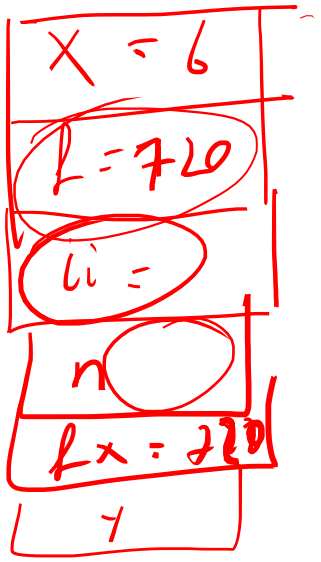
```
>> m =[1,2,3,4
       5,6,7,8
       9,10,11,12]
m =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> trasposta(m)
ans =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

E se usassimo gli script?

E se usassimo uno script file?

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
```

n = x;
fattoriale
fx = f

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

fattoriale.m

script

n = y
fattoriale
fy = f

disp(ii)

Limiti degli script-files

Problemi:

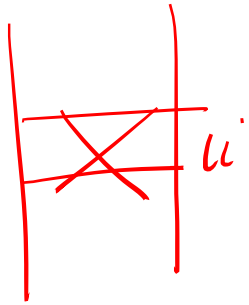
- Come fornisco l'input allo script?
- Dove recupero l'output?

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
```

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

fattoriale.m



Limiti degli script-files

Problemi:

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

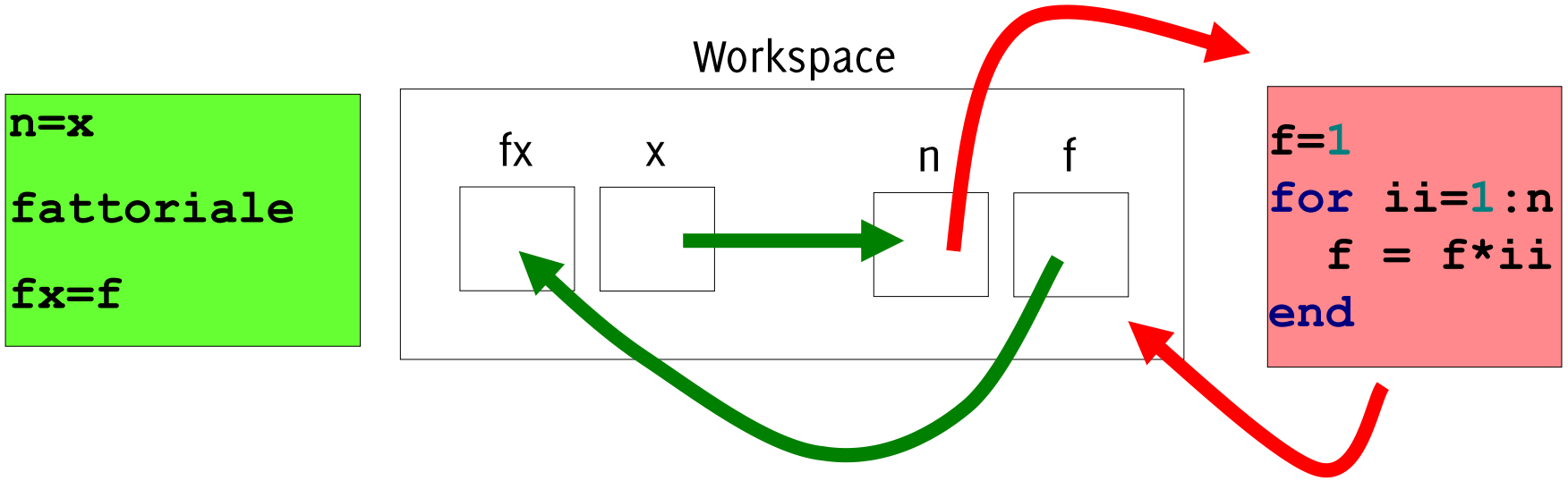
Gli script utilizzano le variabili del workspace:

```
x = input('inserisci x: ');
n=x                    ← Prepara l'input in n
fattoriale             ← chiama lo script
fx=f                   ← Salva il risultato in f
if (fx>220)
  y = input('inserisci y: ');
  n=y                  ← Prepara l'input
  fattoriale           ← chiama lo script
  fy=f                 ← Salva il risultato in f
end
```

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

fattoriale.m

Limiti degli script-files (2)



- 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)
- Le funzioni non hanno questi problemi

Esempi

Esercizio

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

La successione di Fibonacci è definita così:

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

Esercizio

Scrivere un programma che chiede all'utente di inserire un numero positivo n (nel caso in cui il numero non è positivo ripetere inserimento) e verifica se questo è perfetto

Se n non è perfetto dice se è abbondante o difettivo e richiede un secondo numero intero positivo m e controlla se n ed m sono amici. Si stampa a schermo il risultato di questo controllo.

Un numero è perfetto se corrisponde alla somma dei suoi divisori, escluso se stesso (es. 6 è perfetto 1 + 2 +3 = 6)
Un numero è abbondante se è > della somma dei suoi divisori (es 15 è abbondante 1 + 3 +5 < 15), altrimenti difettivo (es 12 è difettivo, 1+2+3+4+6 > 12)
Due numeri a,b sono amici (o amicabili) se la somma dei divisori di a è uguale a b e viceversa (es 220 e 284)

Funzioni Built In

Alcune funzioni built in

A(n,n) = 0
A = zeros(n)

Funzione	Significato
zeros (n)	Restituisce una matrice n×n di zeri
zeros (m,n)	Restituisce una matrice m×n di zeri
zeros (size(arr))	Restituisce una matrice di zeri della stessa dimensione di arr
ones(n)	Restituisce una matrice n×n di uno
ones(m,n)	Restituisce una matrice m×n di uno
ones(size(arr))	Restituisce una matrice di uno della stessa dimensione di arr
eye(n)	Restituisce la matrice identità n×n
eye(m,n)	Restituisce la matrice identità m×n
length(arr)	Ritorna la dimensione più lunga del vettore
size(arr)	Ritorna un vettore [r c] con il numero r di righe e c di colonne della matrice; se **`arr`** ha più dimensioni ritorna array con numero elementi per ogni dimensione

Funzioni predefinite

Esempi

- a = zeros(2); $\longrightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
- b = zeros(2,3); $\longrightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- c = [1 2; 3 4];
- d = zeros(size(c)); $\longrightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Funzioni Aritmetiche

Funzione	Scopo
ceil(x)	approssima x all'intero immediatamente maggiore
floor(x)	approssima x all'intero immediatamente minore
fix(x)	approssima x all'intero più vicino verso lo zero
[m,pos] = max(x)	se x è un vettore, ritorna il valore massimo in x e, opzionalmente, la collocazione di questo valore in x; se x è matrice, ritorna il vettore dei massimi delle sue colonne
[m,pos] = min(x)	se x è un vettore, ritorna il valore minimo nel vettore x e, opzionalmente, la collocazione di questo valore nel vettore; se x è matrice, ritorna il vettore dei minimi delle sue colonne
mean(x)	se x è un vettore ritorna uno scalare uguale alla media dei valori di x; se x è una matrice, ritorna il vettore contentente le medie dei vettori colonna di x;
mod(m,n)	mod(m,n) è m – q .* n dove q = floor(m ./ n) se n ~= 0
round(x)	approssima x all'intero più vicino
rand(N)	Restituisce una matrice di NxN numeri casuali con distribuzione uniforme tra 0,1

Quindi questa funzione….

```
function [minore, maggiore] = minmax(a,b,c)
minore = a;
maggiore = a;
if minore < b
    minore = b;
end
if maggiore > b
    maggiore = b;
end

if minore < c
    minore = c;
end
if maggiore > c
    maggiore = c;
end
```

Si poteva scrivere così

```
function [minore, maggiore] = minmax(a,b,c)
  minore = min([a,b,c]);
  maggiore = max([a,b,c]);
```

funzioni `min` (e anche `max`) applicate a vettori e matrici

```
>> b = [4 7 2 6 5]
b =  4     7     2     6     5
>> min(b)
ans = 2
>> [x y]=min(b)
x =  2
y =  3
>>
```

(con un solo risultato) dà il valore del minimo

con due risultati dà anche la posizione del minimo

```
>> a = [24 28 21; 32 25 27; 30 33 31; 22 29 26]
a = 24    28    21
    32    25    27
    30    33    31
    22    29    26
>> min(a)
ans = 22    25    21

>> [x y]=min(a)
x =  22    25    21
y =  4     2     1
>>
```

per una matrice dà vettore dei minimi nelle colonne

per una matrice, con due risultati dà due vettori dei valori minimi nelle colonne e della loro posizione (riga)

Funzioni Aritmetiche

Prod(vettore) calcola il prodotto di tutti gli elementi di vettore

Esempio: alternativa «alla Matlab» per il calcolo del fattoriale

```
function k = fattoriale2(n)
k = prod([n : -1 : 1]);
```

Altre funzioni importanti

`length(v)`, restituisce la lunghezza del vettore

`size(A)` restituisce un vettore contenente le dimensioni dell'array A (come si vedono da whos)

`size(A, dim)` restituisce il numero di elementi di A lungo la dimensione dim

ATTENZIONE che **`length`** su matrici restituisce la dimensione maggiore! O meglio restituisce **`max(size(A))`**