POLITECNICO DI MILANO

Matlab: Funzioni

Informatica B

Daniele Loiacono

Funzioni

A cosa servono le funzioni?


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

A cosa servono le funzioni?


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

A cosa servono le funzioni?

- Riusabilità
  - Scrivo una sola volta codice utilizzato spesso
  - Modifiche e correzioni sono gestibili facilmente
- Leggibilità
  - Incapsulo porzioni di codice complesso
  - Aumento il livello di astrazione dei miei programmi
- Flessibilià
  - Posso aggiungere funzionalità non presenti nelle funzioni di libreria

Usiamo uno script file?

- Uno script file può essere usato per incapsulare prozioni di codice riusabili in futuro

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

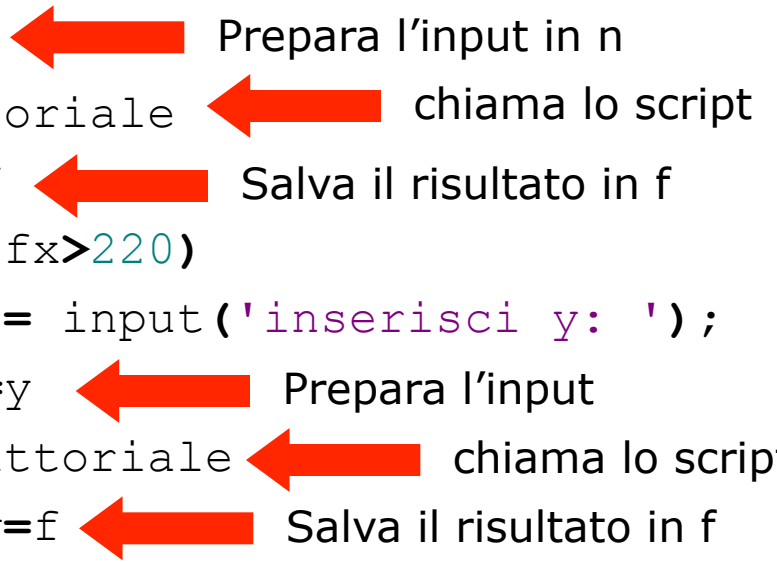
```
f=1
for i=1:n
   f = f*i
end
```

fattoriale.m

Limiti degli script-files

- Problemi:
  - Come fornisco l'input allo script?
  - Dove recupero l'output?
- Gli script utilizzano le variabili dello workspace:

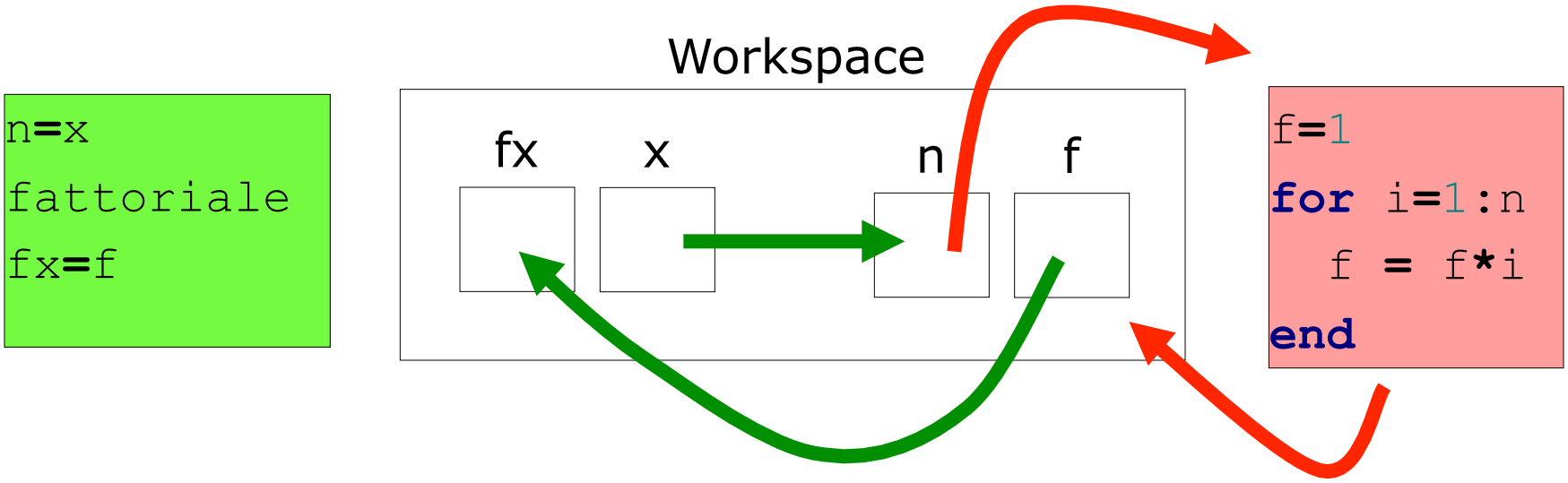
```
x = input('inserisci x: ');
n=x
fattoriale
fx=f
if (fx>220)
   y = input('inserisci y: ');
   n=y
   fattoriale
   fy=f
end
```




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

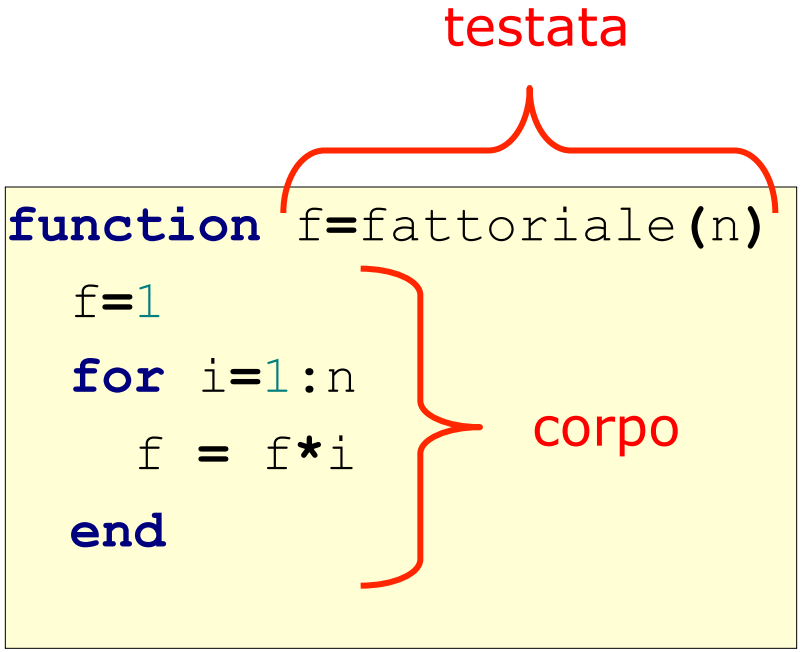
fattoriale.m

Limiti degli script-files (2)



- Questo meccanismo ha molti svantaggi:
  - poco leggibile
  - richiede molte istruzioni
  - poco sicuro

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)

- La testata 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, possiamo usare un array:

  function [v1,v2,…] = f(x,y)

- Esempio:

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

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
```

I parametri

- Definizioni:
  - I parametri formali sono le variabili usate come argomenti e valore di ritorno nella definizione della funzione
  - I parametri attuali sono i valori (o le variabili) usati come argomenti/valore di ritorno nella invocazione della funzione
- Esempio:

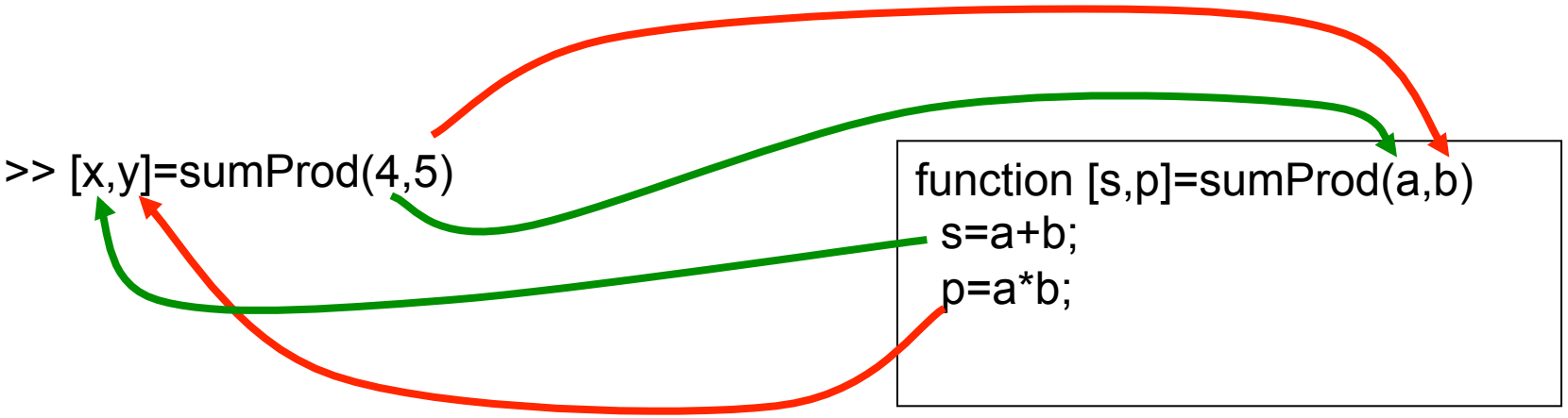
```
function f=fattoriale(n)
   f=1
   for i=1:n
      f = f*i
   end

>> fx = fattoriale(5)
```

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.
- Un invocazione di funzione deve contenere un numero di parametri attuali identico al numero di parametri formali
- 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 di nomi)
  - Quando la funzione viene eseguita, il workspace "locale" viene distrutto!
- Quando viene invocata una funzione:
  - Vengono calcolati i valori dei parametri attuali di ingresso
  - Viene creato un workspace "locale" per la funzione
  - I valori dei parametri attuali di ingresso vengono copiati nei parametri formali all'interno del workspace "locale"
  - Viene eseguita la funzione
  - Vengono copiati i valori di ritorno dal workspace "locale" a quello "principale" (nei corrispondenti parametri attuali)
  - 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')
  x=w+1;        %(4')
```

W "principale" dopo (2)

x=3
w=2

W "principale" dopo (3)

x=3
w=2
r= 8

~~W "locale"dopo (3)~~

W "locale" dopo(1')

x=4
y=8

W "locale" dopo(3')

x=0
y=8
z=4

W "locale" (4')

x=0
y=8
z=4
w=? → errore

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') → 0 se la funzione non esiste

Esempi


```
function [pres, pos]=cerca(x, v)
    p=0; pos=[];
    for i=1:length(v)
        if v(i)==x
            p=p+1;
            pos(p)=i;
        end
    end
    pres=p>0;
```

```
>> A=[1, 2, 3, 4, 3, 4, 5, 4, 5, 6]
A = 1 2 3 4 3 4 5 4 5 6
>> [p, i]=cerca(4,A)
p = 1
i = 4 6 8
```

- Esercizio: implementare usando find()

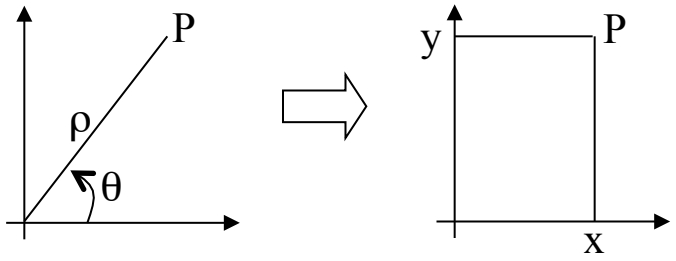
Esempi (2)


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

Esempi (3)


```
>> P.ro=1; P.teta=pi/3
P =
      ro: 1
    teta: 1.0472
>> Q=polar2cart(P)
Q =
    x: 0.5000
    y: 0.8660

>> R.to=1; R.teta=pi/3
R =
      to: 1
    teta: 1.0472
>> Q=polar2cart(R)
??? Reference to non-existent
field 'ro'.
Error in ==> polar2cart at 2
    c.x=p.ro*cos(p.teta);
```




```
function [c]=polar2cart(p)
    c.x=p.ro*cos(p.teta);
    c.y=p.ro*sin(p.teta);
```

Ricorsione

Ricorsione

- Che cos'è la ricorsione?
  - Un sottoprogramma P richiama se stesso (ricorsione diretta)
  - Un sottoprogramma P richiama un'altro sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)
- A cosa serve?
  - È una tecnica di programmazione molto potente
  - Permette di risolvere in maniera elegante problemi complessi

Programmazione ricorsiva

- Per risolvere un problema attraverso la programmazione ricorsiva sono necessari alcuni elementi
  - **Caso base**: caso elementare del problema che può essere risolto immediatamente
  - **Passo ricorsivo**: chiamata ricorsiva per risolvere uno o più problemi più semplici
  - **Costruzione della soluzione**: costruzione della soluzione sulla base del risultato delle chiamate ricorsive

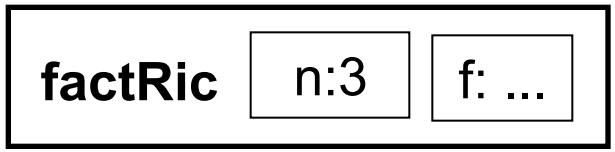
Esempio: il fattoriale

- Definizione:
  f(n) = n! = n*(n-1)*(n-2)*…*3*2*1
- Passo ricorsivo:
  f(n) = n*f(n-1)
- Caso base:
  f(0)=1

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

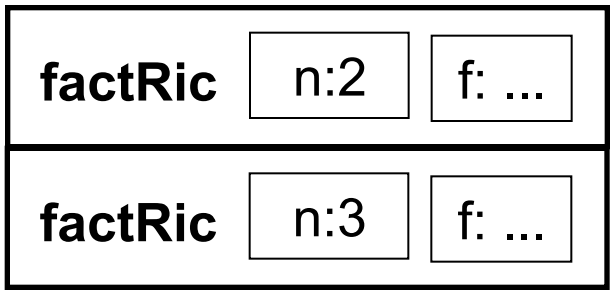
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



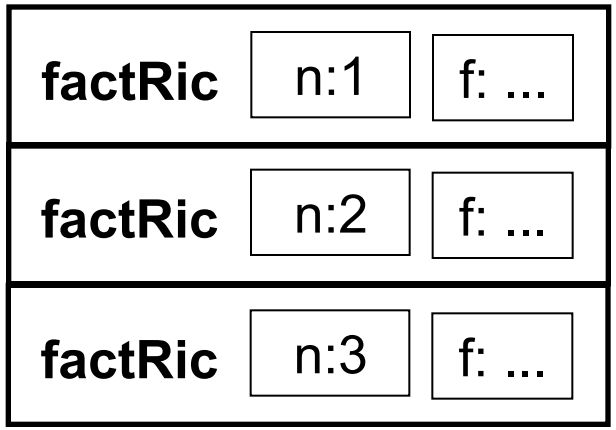
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



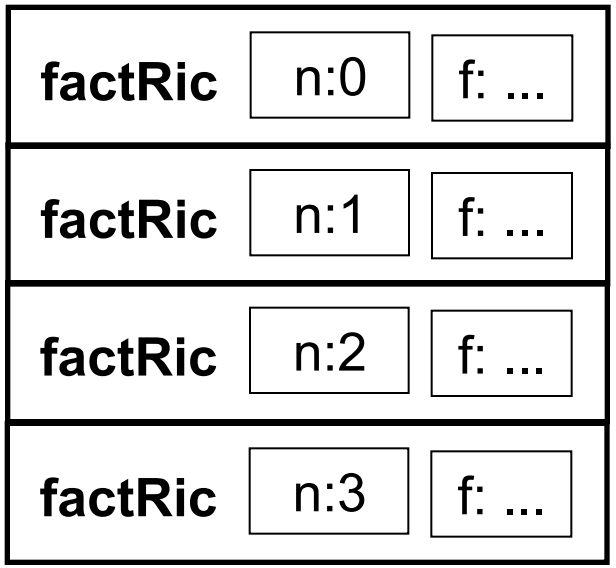
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



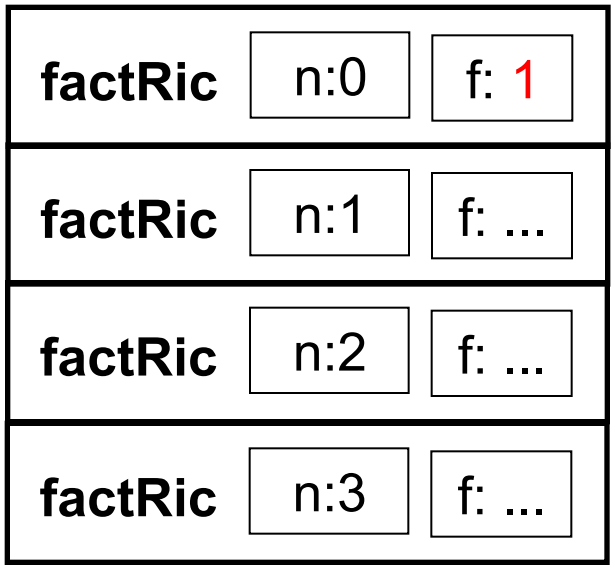
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



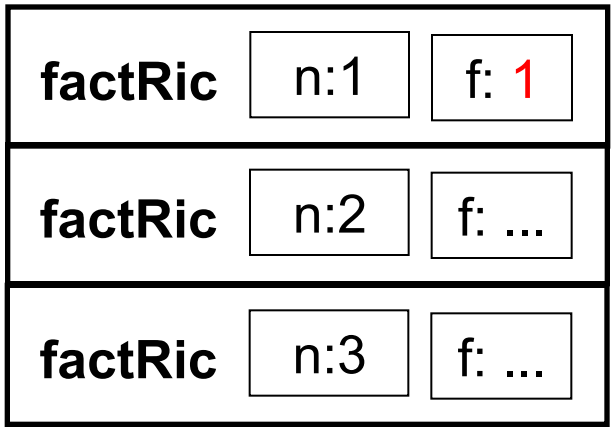
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



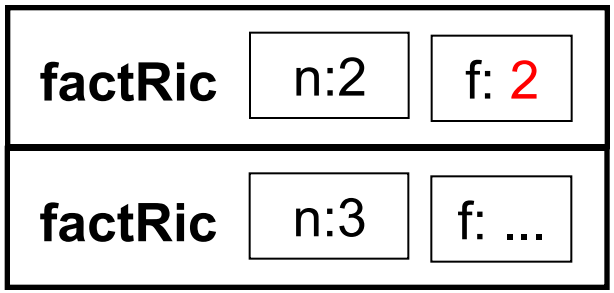
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



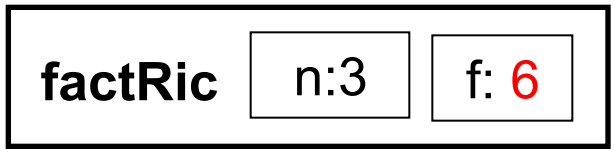
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



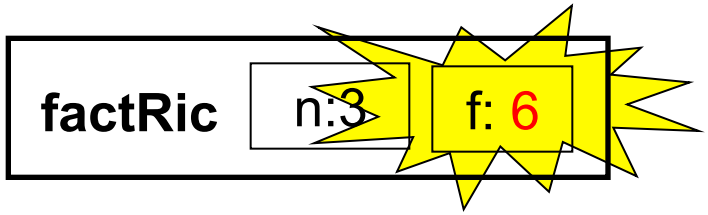
Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



Funzionamento ricorsione


```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```



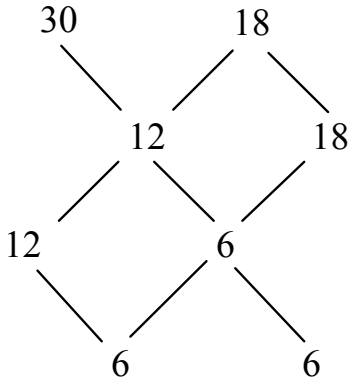
Esempio: Fibonacci

- È una sequenza di numeri interi in cui ogni numero si ottiene sommando i due precedenti nella sequenza. I primi due numeri della sequenza sono per definizione pari ad 1.
  - $f_1 = 1$ (caso base)
  - $f_2 = 1$ (caso base)
  - $f_n = f_{n-1} + f_{n-2}$ (passo ricorsivo)

```
function [fib]=FiboRic(n)
  if n==1 | n==2
      fib=1;
  else
      fib=FiboRic(n-2)+FiboRic(n-1);
  end
```

Esempio: MCD

- Algoritmo di Euclide
  - se m = n, MCD(m,n) = m (caso base)
  - se m > n, MCD(m,n) = MCD(m-n,n) (caso risorsivo)
  - se m < n, MCD(m,n) = MCD(m,n-m) (caso risorsivo)
- Esempio: MCD(30,18)



Esempio: MCD

- Algoritmo di Euclide
  - se m = n, MCD(m,n) = m (caso base)
  - se m > n, MCD(m,n) = MCD(m-n,n) (caso risorsivo)
  - se m < n, MCD(m,n) = MCD(m,n-m) (caso risorsivo)
- Implementazione

```
function [M]=MCDeuclidRic(m,n)
  if m==n
      M=m;
  else
      if m>n
          M = MCDeuclidRic(m-n,n);
      else
          M = MCDeuclidRic(m,n-m);
      end
  end
```

Problemi nell'uso della ricorsione

- Terminazione della catena ricorsiva
  - È presente il caso base?
  - Viene raggiunto sempre dalla catena di chiamate ricorsive?
  - Esempi

```
function [f]=factRic(n)
     f=n*factRic(n-1);
```

Catena infinita di chiamate con argomento decrescente

```
function [f]=factRic(n)
     …factRic(n);
```

Catena infinita di chiamate identiche

Problemi nell'uso della ricorsione

- Uso della memoria
  - La programmazione ricorsiva comporta spesso un uso inefficiente della memoria per la gestione degli spazi di lavoro delle chiamate generate
  - In alcuni casi viene comunque preferita ad altri approcci per la sua eleganza e semplicità
  - In altri casi, si può ricorrere ad implementazioni iterative
  - Esempio

```
function [fl]=Fiblist(n)
  fl(1)=1;
  fl(2)=1;
  for k=3:n
      fl(k)=fl(k-2)+fl(k-1);
  end
```

Funzione iterativa che calcola i primi n numeri di fibonacci

Variabili funzione

Variabili funzioni

- Matlab permette di assegnare a variabili valori di tipo "funzione"
- Un valore di tipo funzione può essere assegnato a una variabile (quindi passarlo come parametro), detta handle
- L'handle può essere applicato a opportuni argomenti per ottenere una invocazione della funzione

Assegnamento di un valore di tipo funzione

- Handle di una funzione esistente

  f = @nome_funzione

  - Esempio

```
>> seno=@sin
seno = @sin
>> seno(pi/2)
ans = 1
```

- Handle di una funzione definita ex-novo

  f = @(x,y…)<expr>

  - x,y,… sono i parametri della funzione
  - <expr> è un'espressione che calcola il valore della funzione
  - Esempio

```
>> sq=@(x)x^2
sq = @(x)x^2
>> sq(8)
ans =  64
```

Funzioni di ordine superiore

- Se un parametro di una funzione **f** è un handle (cioè contiene un valore di tipo funzione) allora **f** è una **funzione di ordine superiore**
- L'handle passato come parametro consente ad **f** di invocare la funziona passata
- Esempio: funzione map che applica una funzione f a tutti gli elementi conetenuti nel parametro vin e ritorna i risultati in vout

```
function [vout]=map(f, vin)
    for i=1:length(vin)
        vout(i)=f(vin(i));
    end;
```

handle

Invoca la funzione passata come argomento

```
>> A=[1,2,3,4,5,6];
>> map(sq,A)
ans = 1  4  9  16  25  36
```

Esempio: funzione accumulatore

- Funzione accumulatore: `[x] =acc(f, a, u)`
  - applica cumulativamente l'operazione binaria f (con elemento neutro u) a tutti gli elementi di a:

f(…f(f(f(u,a(1)),a(2)),a(3))…,a(length(a)))

```
function [x]=acc(f, a, u)
    x=u;
    for i=1:length(a)
        x=f(x, a(i));
    end
```

- Funzione sommatoria: `function [s]=sommatoria(a)`
  - calcola la sommatoria degli elementi di a

```
function [s]=sommatoria(a)
    som=@(x,y)x+y;
    s=acc(som,a,0);
```