ricorsione

funzioni ricorsive

definizioni

- un oggetto si dice ***ricorsivo*** se è definito totalmente o parzialmente in termini di ***se stesso***
- la ricorsione è un mezzo molto potente per le definizioni e le dimostrazioni matematiche (***induzione***)
- si usano algoritmi ricorsivi quando il problema da risolvere presenta caratteristiche proprie di ricorsività (può essere risolto in termini di uno o più problemi analoghi ma di dimensioni inferiori)

immagine ricorsiva

A. Ferrari

definizioni ricorsive

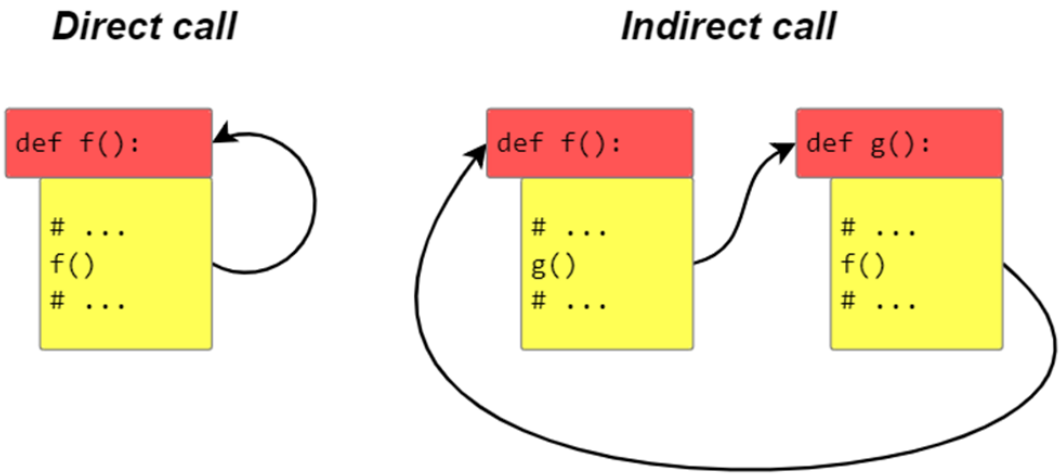
- definizione dei ***numeri naturali***:
  1) 1 è un numero naturale
  2) il successore di un numero naturale è un numero naturale
- definizione di ***fattoriale*** di un numero intero positivo:
  1) 0! = 1
  2) n! = n * (n−1)!
- calcolo del ***MCD*** tra due numeri A e B (A>B)
  algoritmo di Euclide
  1) dividere A per B
  2) se il resto R è zero
     allora MCD(A,B)=B
     altrimenti MCD(A,B)=MCD(B,R)

```
def mcd(a: int, b:int) -> int:
    r = a % b
    if (r==0):
        return b    #condizione di terminazione
    else:
        return(mcd(b,r))
```

- il potere della ricorsività consiste nella possibilità di definire un insieme anche infinito di oggetti con un numero finito di comandi
- il problema principale quando si usano algoritmi ricorsivi è quello di garantire una ***terminazione*** (caso terminale, condizione di ***fine***, condizione iniziale)
- non è sufficiente inserire una condizione di terminazione, ma è necessario che le chiamate ricorsive siano tali da determinare il ***verificarsi*** di tale condizione in un numero finito di passi

procedure e funzioni ricorsive

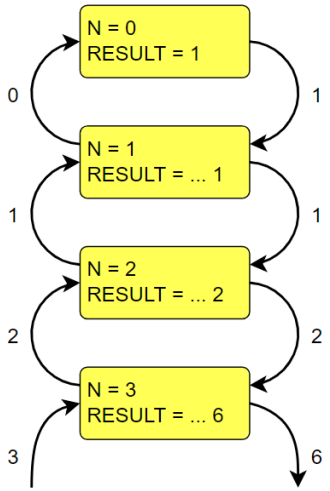
- un sottoprogramma ricorsivo è una procedura (o ***funzione***) all'interno della quale è presente una ***chiamata a se stessa*** o ad altro sottoprogramma che la richiama
- la ricorsione è ***diretta*** se la chiamata è interna al sottoprogramma altrimenti si dice indiretta
- molti linguaggi consentono ad una funzione (o procedura) di chiamare se stessa



fattoriale: ricorsione

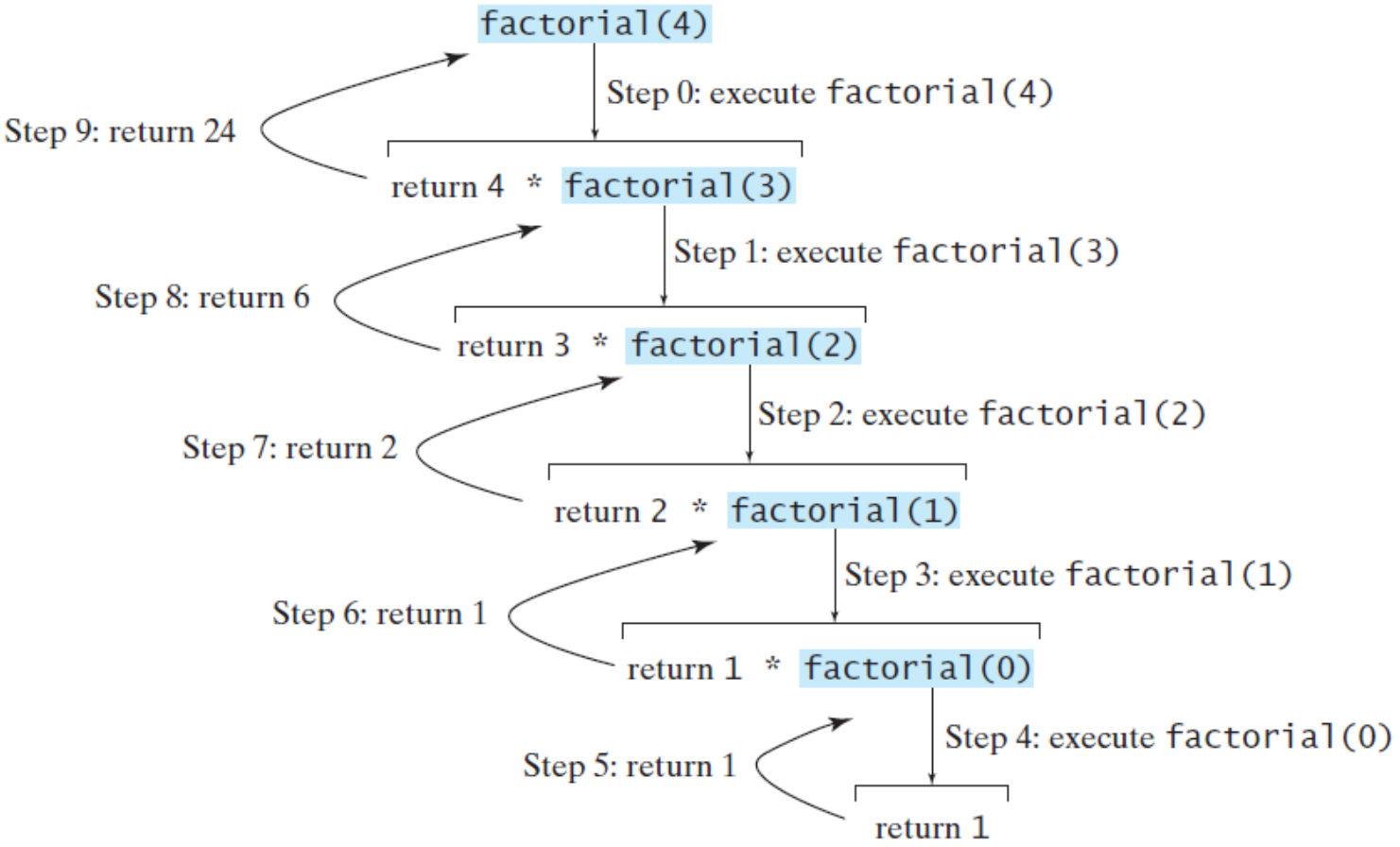
- ad ogni invocazione di una funzione, viene creato nello **stack** un nuovo record
- **contesto locale** alla particolare attivazione della funzione stessa

```
def factorial(n: int) -> int:
    result = 1
    if n > 1:
        result = n * factorial(n - 1)
    return result
```



Ai primordi (Fortran 66 ecc.) solo allocazione statica
Spazio fisso ed unico per dati locali ad una funzione
→ no ricorsione

esecuzione


factorial(4)
Step 0: execute factorial(4)
Step 9: return 24
return 4 * factorial(3)
Step 1: execute factorial(3)
Step 8: return 6
return 3 * factorial(2)
Step 2: execute factorial(2)
Step 7: return 2
return 2 * factorial(1)
Step 3: execute factorial(1)
Step 6: return 1
return 1 * factorial(0)
Step 4: execute factorial(0)
Step 5: return 1
return 1

domande ed esercizi

- cosa si intende con funzione ricorsiva?
- utilizzando la funzione precedentemente definite quante volte viene richiamata la funzione se si calcola il fattoriale di 6?
- dai la definizione matematica ricorsiva della funzione $f(n) = 2^n$ con n positivo
- dai la definizione matematica ricorsiva di $f(n) = 0 + 1 + 2 + \ldots n$ con n positivo
- scrivi una funzione con ricorsività infinita
- cosa si intende con ricorsione diretta?
- cosa si intende con ricorsione indiretta?

istanziazione

- ogni nuova chiamata di un sottoprogramma ricorsivo determina una ***nuova istanza*** dell'ambiente locale (distinto da quello precedente che comunque resta attivo)
- ad ogni chiamata si alloca ***nuova memoria*** e questo può determinare problemi di spazio
- i vari ambienti vengono salvati in una struttura di tipo ***LIFO*** (Stack o Pila) in modo che alla terminazione di una determinata istanza venga riattivata quella immediatamente precedente e così via

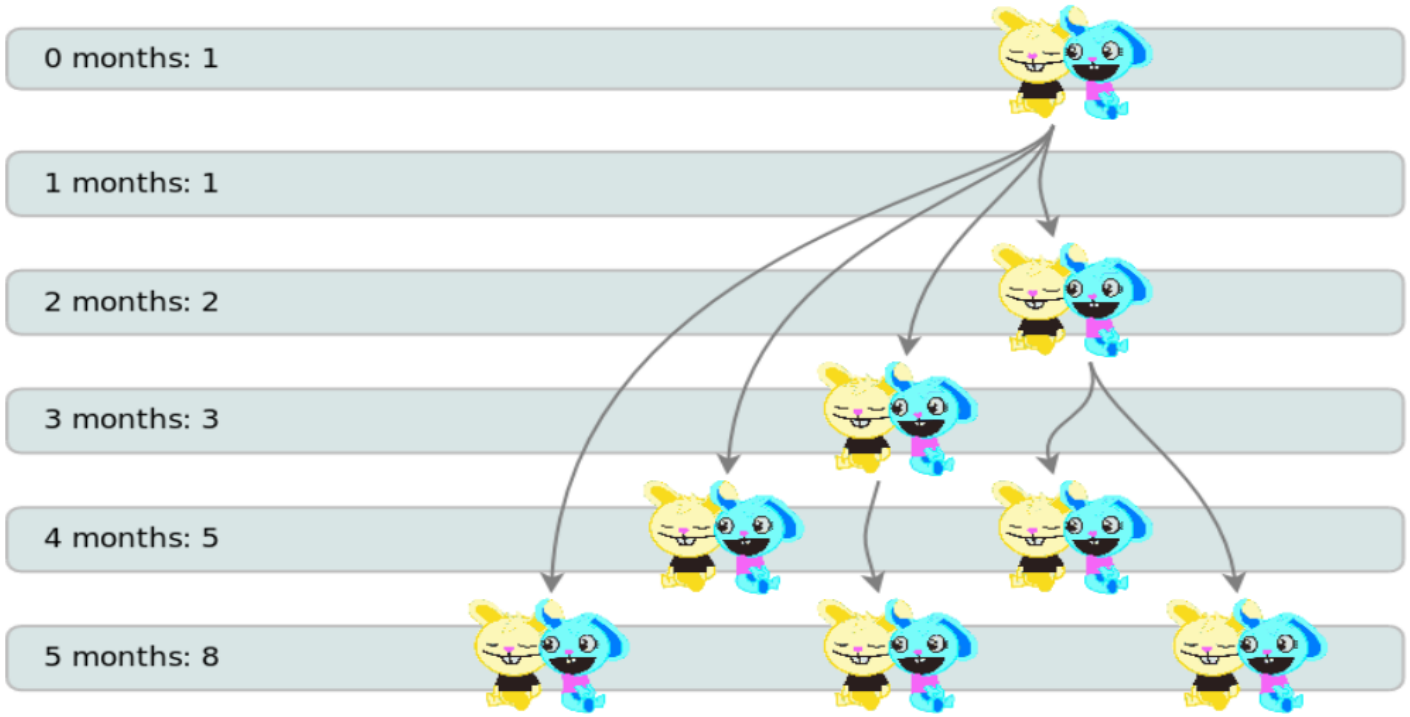
stack dell'applicazione

- ***pila***: memoria dinamica LIFO (Last In First Out)
  - dimensione massima prefissata
- il programma ci memorizza automaticar
  - ***indirizzo*** di ritorno per la funzione
    - inserito alla chiamata, estratto all'uscita
  - ***parametri*** della funzione
    - inseriti alla chiamata, eliminati all'uscita
  - ***variabili locali***, definite nella funzione
    - eliminate fuori dall'ambito di visibilità

i conigli di Fibonacci

A. Ferrari

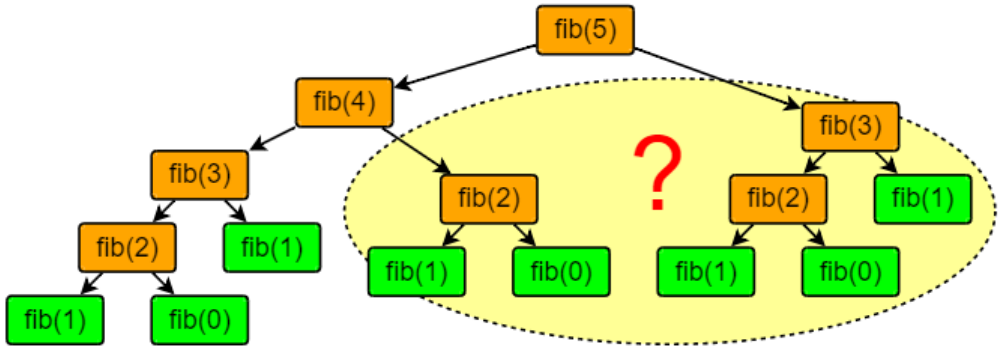
fib(0) = fib(1) = 1;
fib(n) = fib(n-1) + fib(n-2);



Fibonacci ricorsione

A. Ferrari

```
def fibonacci(n: int) -> int:
    result = 1
    if n > 1:
        result = fibonacci(n-1) + fibonacci(n-2)
    return result
```



Fibonacci iterativo

A. Ferrari

```
def fibonacci(n: int) -> int:
    valore = 1
    precedente = 0

    for i in range(n):
        valore, precedente = valore + precedente, valore

    return valore
```

domande ed esercizi

A. Ferrari

```
def f(n):
    if n > 0:
        print(n % 10)
        f(n // 10)

def f2(n):
    if n > 0:
        f2(n // 10)
        print(n % 10)

print(f(12345))
print(f2(12345))
```

- qual è l'output del programma?
- scrivere la docstring di entrambe le funzioni

caratteristiche di una funzione ricorsiva

A. Ferrari

- all'interno della funzione è presente una istruzione ***if*** che verifica i casi di ***terminazione*** (non chiamata ricorsiva)
- ogni chiamata ricorsiva ***riduce*** il problema
  - lo '***avvicina***' ai casi di terminazione
- per risolvere un problema in modo ricorsivo lo si scompone in ***sottoproblemi dello stesso tipo*** ma di 'dimensione' inferiore

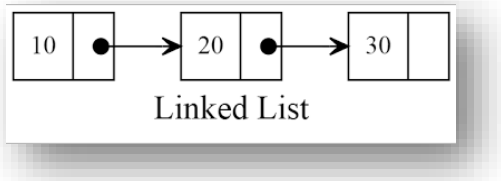
domande ed esercizi

A. Ferrari

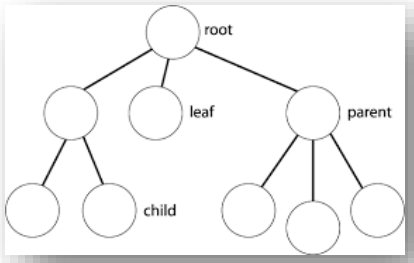
```
def f(n):
    if n > 0:
        print(n, end='-')
        f(n-1)

def f2(n):
    if n > 0:
        f2(n-1)
        print(n, end='-')

f(5)
print()
f2(5)
```

- qual è l'output del programma?
- scrivere una funzione ricorsiva che restituisce la somma delle cifre di un numero intero positivo

- in un ***tipo di dato ricorsivo*** un valore può contenere valori dello stesso tipo
- ***lista*** collegata (linked list)
  - vuota, oppure...
    - nodo di testa, seguito da una lista collegata



- ***albero***
  - vuoto, oppure...
  - nodo di testa, seguito da più alberi



ricorsione

esercizi

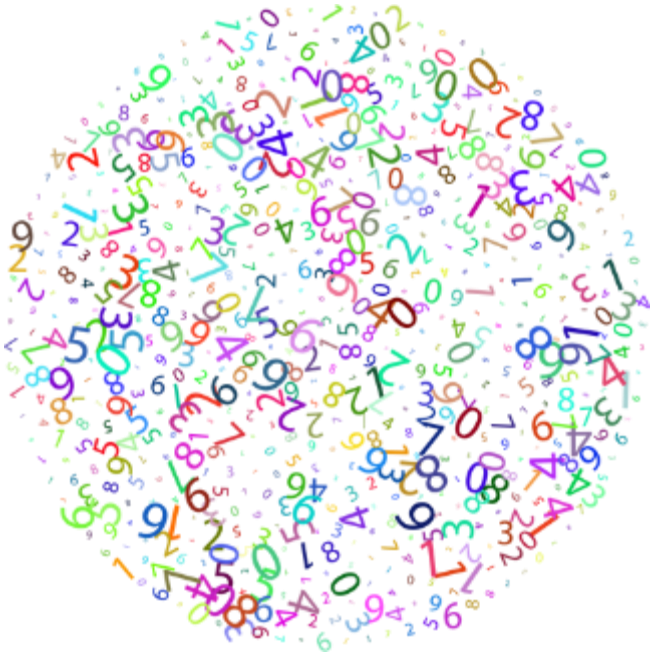
- ***palindromo***
  - *palindromo: testo che rimane uguale se letto al contrario*
  - scrivere una funzione ***ricorsiva*** per riconoscere i palindromi
    - parametro: testo da controllare
    - risultato: bool

stringa palindroma: se ha lunghezza 0 o 1, oppure…
prima lettera == ultima lettera e…
stringa rimanente (senza prima e ultima lettera) palindroma

esercizi (2)

- scrivere una funzione ricorsiva che, data una lista di interi e un valore intero n, restituisce True se almeno uno dei valori della lista è multiplo di n, False altrimenti

considerare il primo elemento della lista poi eventualmente una lista contenente tutti gli elementi meno il primo.
se la lista è vuota …



esercizi (3)

- scrivere una funzione che restituisce la dimensione di una cartella del disco fisso (parametro il nome della cartella)
- **`os.path.isfile(s)`**
  - restituisce **`True`** se **`s`** è il nome di un file
- **`os.path.getsize(f)`**
  - restituisce la dimensione del file **`f`**
- **`os.listdir(c)`**
  - restituisce una lista contente l'elenco dei file e delle sottocartelle di **`c`**