INFORMATICA

RICORSIONE

COSA È LA RICORSIONE?

- Fino ad ora abbiamo visto funzioni che possono chiamare altre funzioni...
- ...ma cosa succede se una funzione si chiama da sola?
- È perfettamente valido ma questo metodo di programmazione prende un nome particolare:
- **Ricorsione**.

PERCHÉ È UTILE?

- Possiamo definire una cosa in termini di sé stessa
- È molto simile a qualcosa che dovreste avere già visto:
- L'**induzione**.
- Vediamo un esempio: il fattoriale

$$n! = \begin{cases} n \times (n-1)! & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases}$$

FATTORIALE

```
def fattoriale(n):
  if (n == 1):
    return 1
  else:
    m = fattoriale(n-1)
    return m*n


# da qualche parte nel codice
x = fattoriale(3)
printf(“3! = “ + str(x))
```

COSA È SUCCESSO? PERCHÉ FUNZIONA?

- Ogni chiamata genera un nuovo record di attivazione…
- …con una sua copia delle variabili locali e dei parametri
- Non continuiamo a chiamare funzioni all'infinito:
  - Abbiamo un **passo ricorsivo** in cui chiamiamo di nuovo la funzione
  - Abbiamo un **caso base** in cui conosciamo il valore della funzione
  - Prima o poi arriviamo al caso base

STRUTTURA BASE DELLA RICORSIONE

```
if caso base:
   return valore_noto
else:
   chiamata ricorsiva
```

Un caso a cui dobbiamo sempre ricondurci dopo
un numero finito di chiamate

Quando capitiamo nel caso base assumiamo di conoscere
(o saper calcolare direttamente) il valore della funzione

Nella chiamata ricorsiva riconduciamo il problema di trovare
il valore della funzione in termini della funzione stessa

E SE NON ARRIVIAMO AL CASO BASE?

`fattoriale(-3)`

Chiama `fattoriale(-4)`

Che chiama `fattoriale(-5)`

Che a sua volta chiama `fattoriale(-6)`

... molte chiamate dopo...

Stack overflow: abbiamo riempito tutto lo spazio disponibile per lo stack

UN ALTRO ESEMPIO: MASSIMO COMUN DIVISORE

L'algoritmo di Euclide può essere espresso nel seguente modo:

$$mcd(x, y) = \begin{cases} mcd(y, x \,\%\, y) & \text{se } y \neq 0 \\ x & \text{se } y = 0 \end{cases}$$

```
def mcd(x, y):
  if y == 0:
    return x
  else:
    return mcd(y, x%y)
```

Caso base

Valore noto

Chiamata ricorsiva

ESEMPI DI DEFINIZIONI RICORSIVE

Numeri di Fibonacci:

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{se } n > 2 \\ 1 & \text{se } n = 0 \text{ o } n = 1 \end{cases}$$

Lunghezza di una lista/array:

$$\text{lunghezza(lista)} = \begin{cases} 0 & \text{se la lista è vuota} \\ 1 + \text{lunghezza(resto della lista)} & \text{altrimenti} \end{cases}$$

RICORSIONE E ITERAZIONE

- Ricorsione e iterazione sono “equivalenti”:
- Per ogni funzione iterativa possiamo costruire una equivalente funzione ricorsiva.
- Per ogni funzione ricorsiva possiamo costruire una equivalente funzione iterativa…
- …nel caso peggiore gestendo “imitando” in modo iterativo lo stack di chiamate.

QUIZ SULLA RICORSIONE

```
def f(a, b):
  if b == 0:
    return a
  else:
    return 1 + f(a, b - 1)
```

Che risultato otteniamo chiamando f(5,4)?

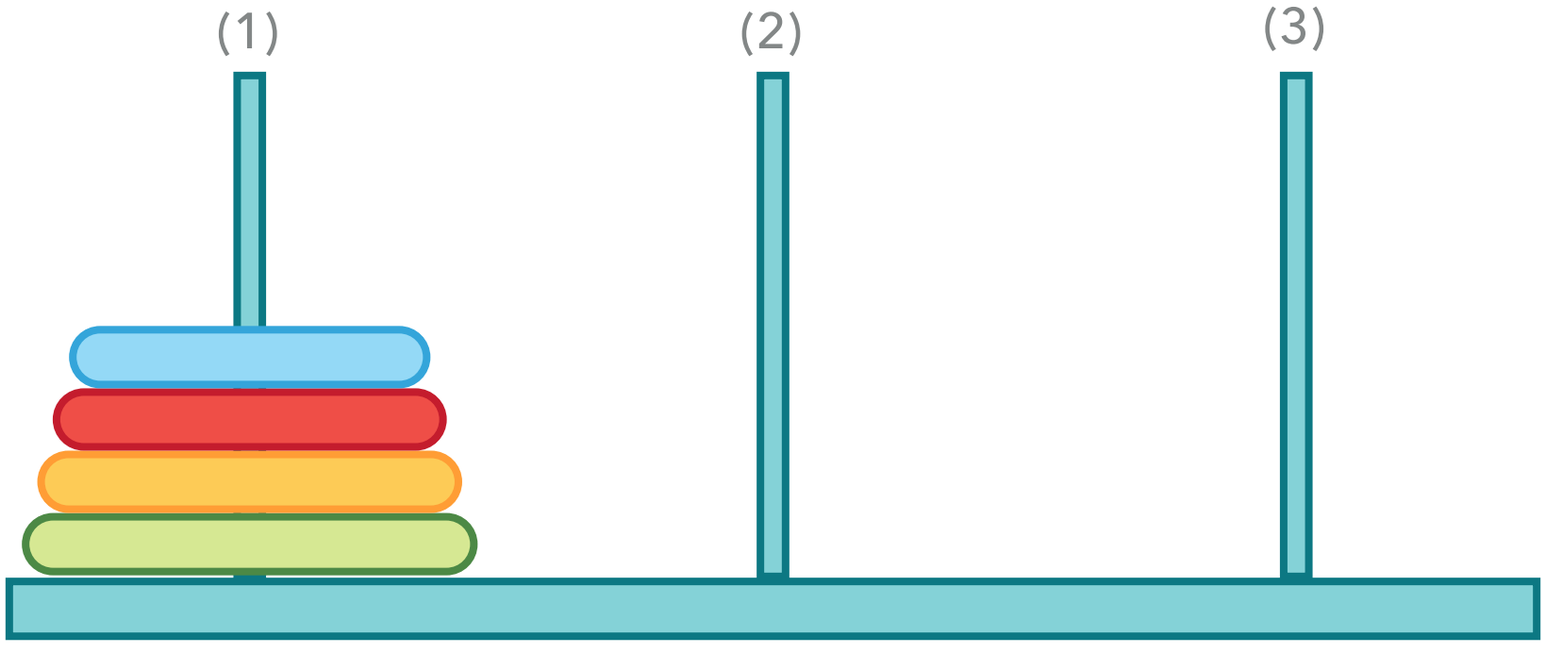
1. Stack overflow
2. 5
3. 9
4. 4

LA TORRE DI HANOI

Spostare tutti i dischi da (1) a (3), eventualmente usando (2)
sotto le seguenti condizioni:

A. Si può spostare un solo disco alla volta

B. Un disco non può essere appoggiato sopra un disco più piccolo

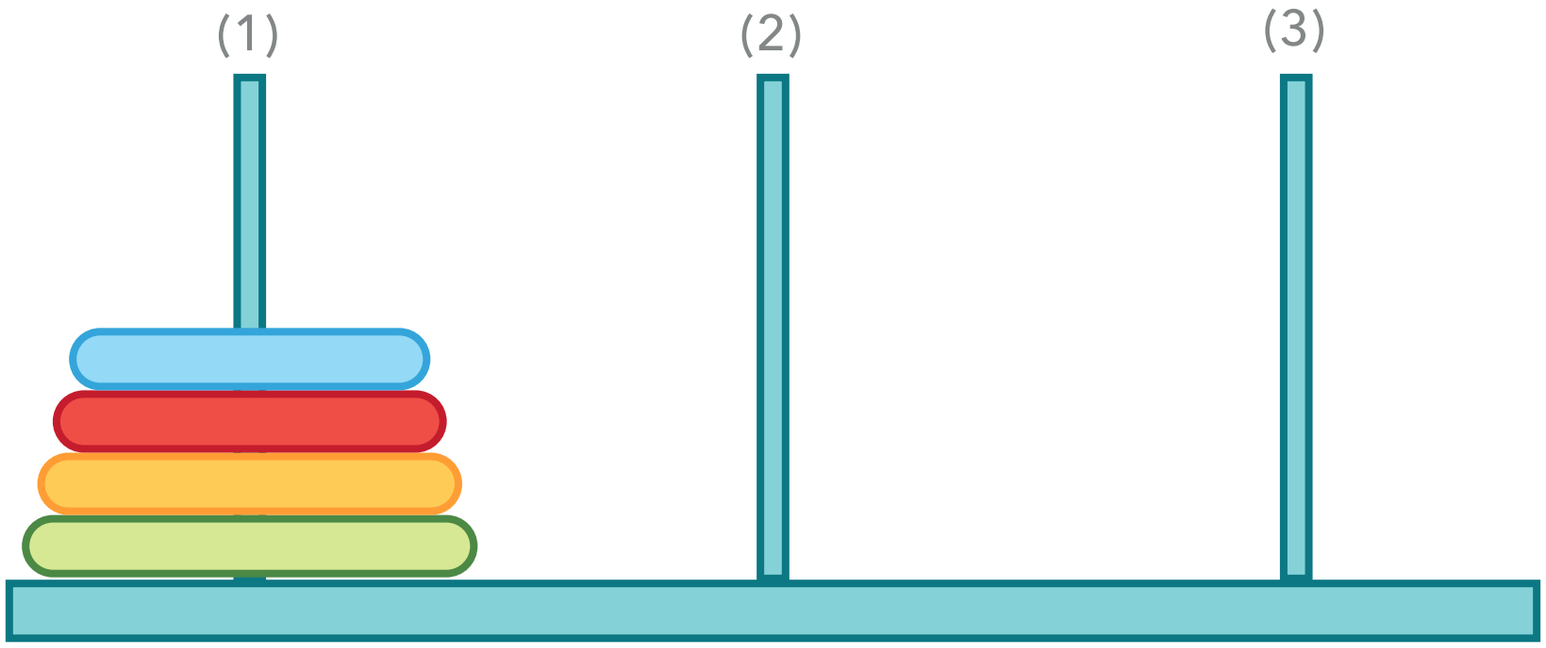


LA TORRE DI HANOI: APPROCCI AL PROBLEMA

Come possiamo ricondurre il problema ad un caso più semplice?

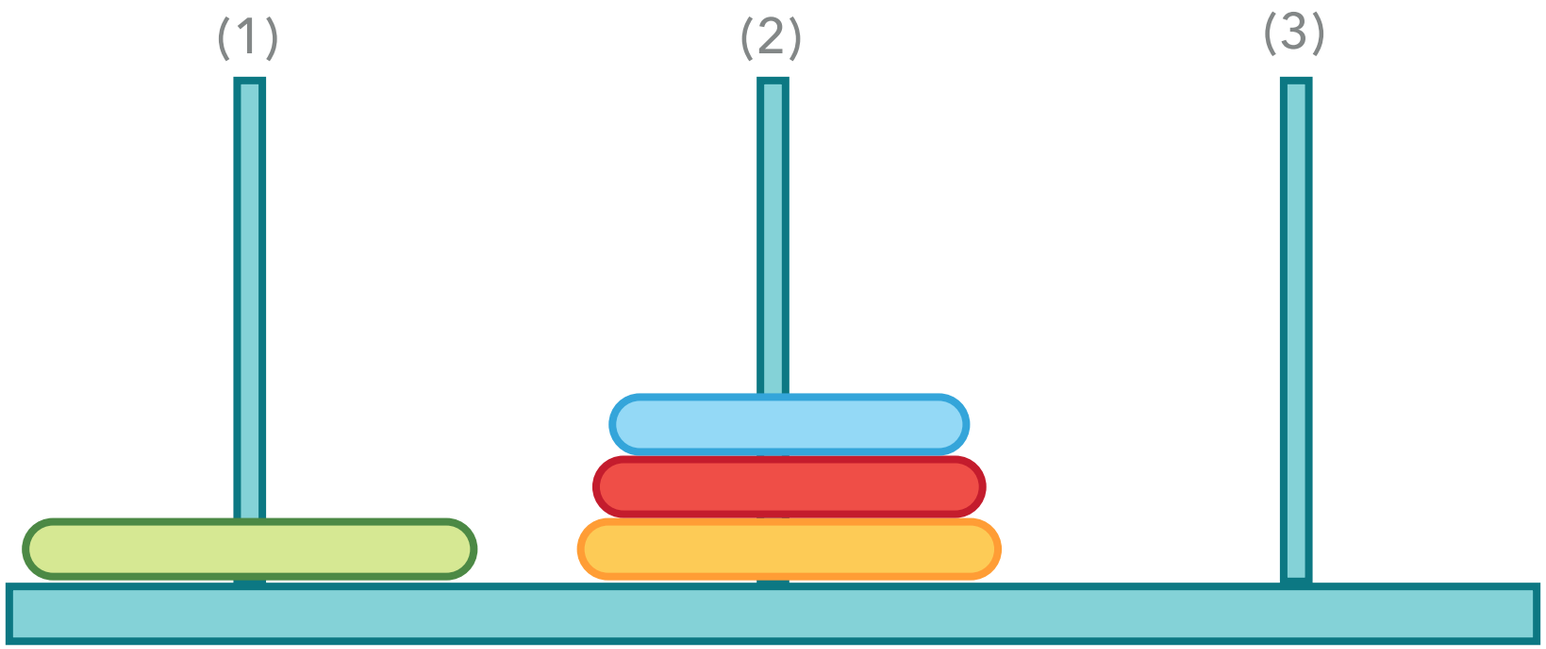
Supponiamo di avere una procedura per spostare n-1 dischi

Come possiamo sfruttarla per risolvere il problema?



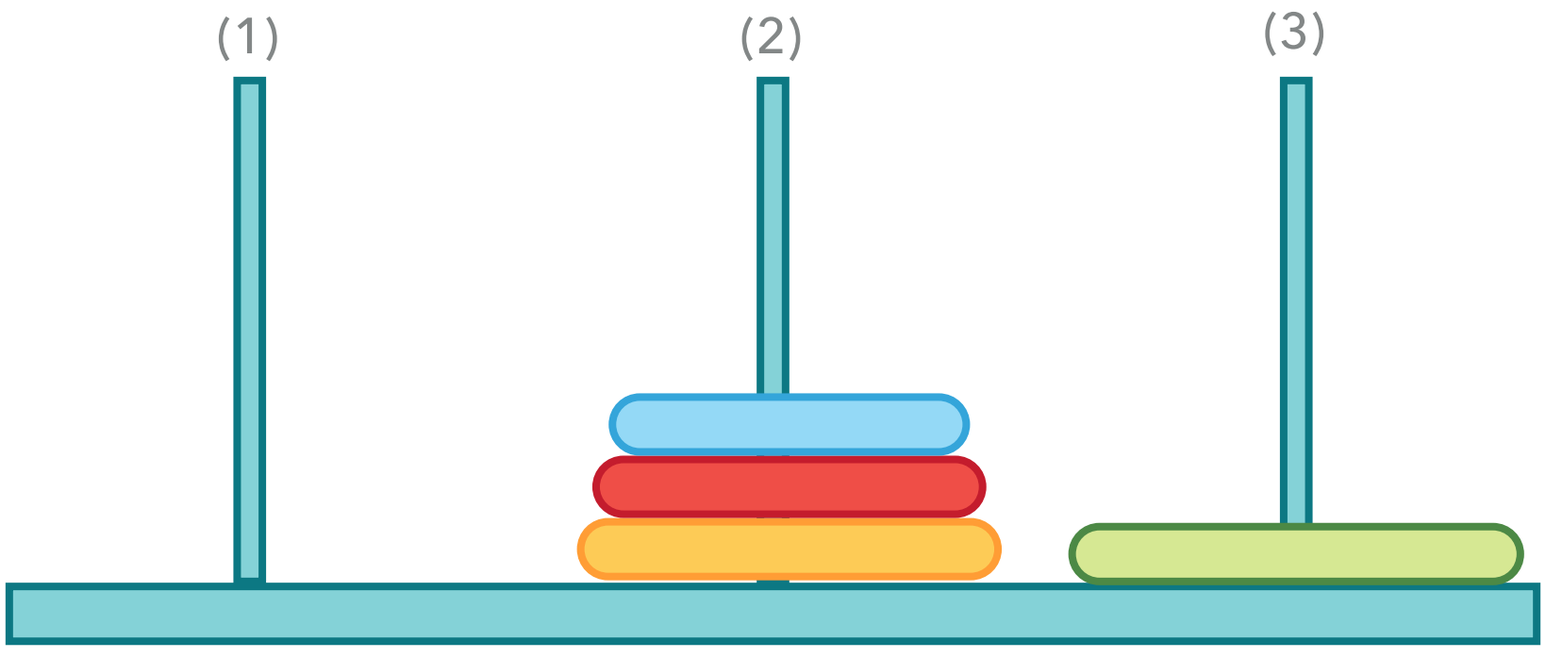
LA TORRE DI HANOI: APPROCCI AL PROBLEMA

Spostiamo n-1 dischi in (2)



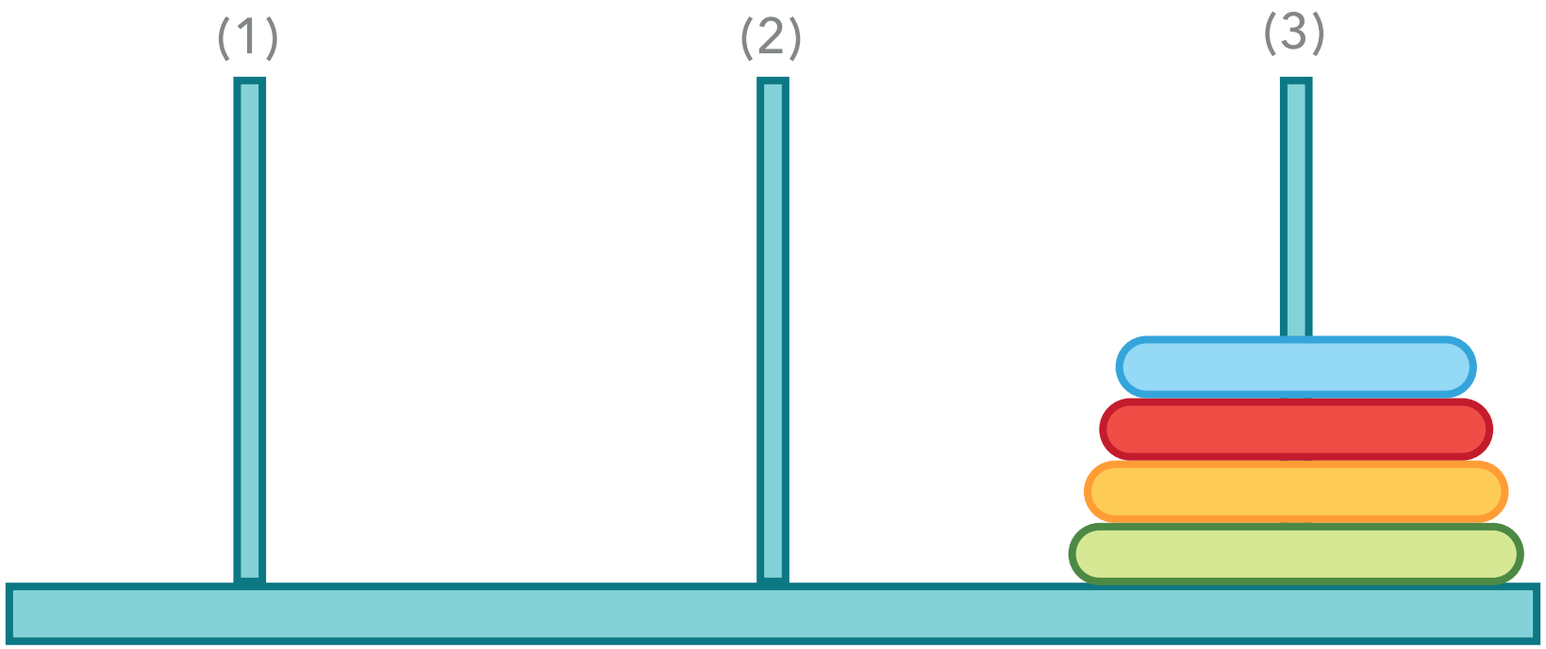
LA TORRE DI HANOI: APPROCCI AL PROBLEMA

Spostiamo il disco più grosso nella destinazione (3)



LA TORRE DI HANOI: APPROCCI AL PROBLEMA

Spostiamo gli n-1 dischi su (3)



LA TORRE DI HANOI

Ma come facciamo a spostare n-1 dischi?

Supponiamo di avere una procedura per spostare n-2 dischi...

... e ci fermiamo quando raggiungiamo 0 dischi: in quel caso non serve fare nulla, dato che non abbiamo dischi da spostare

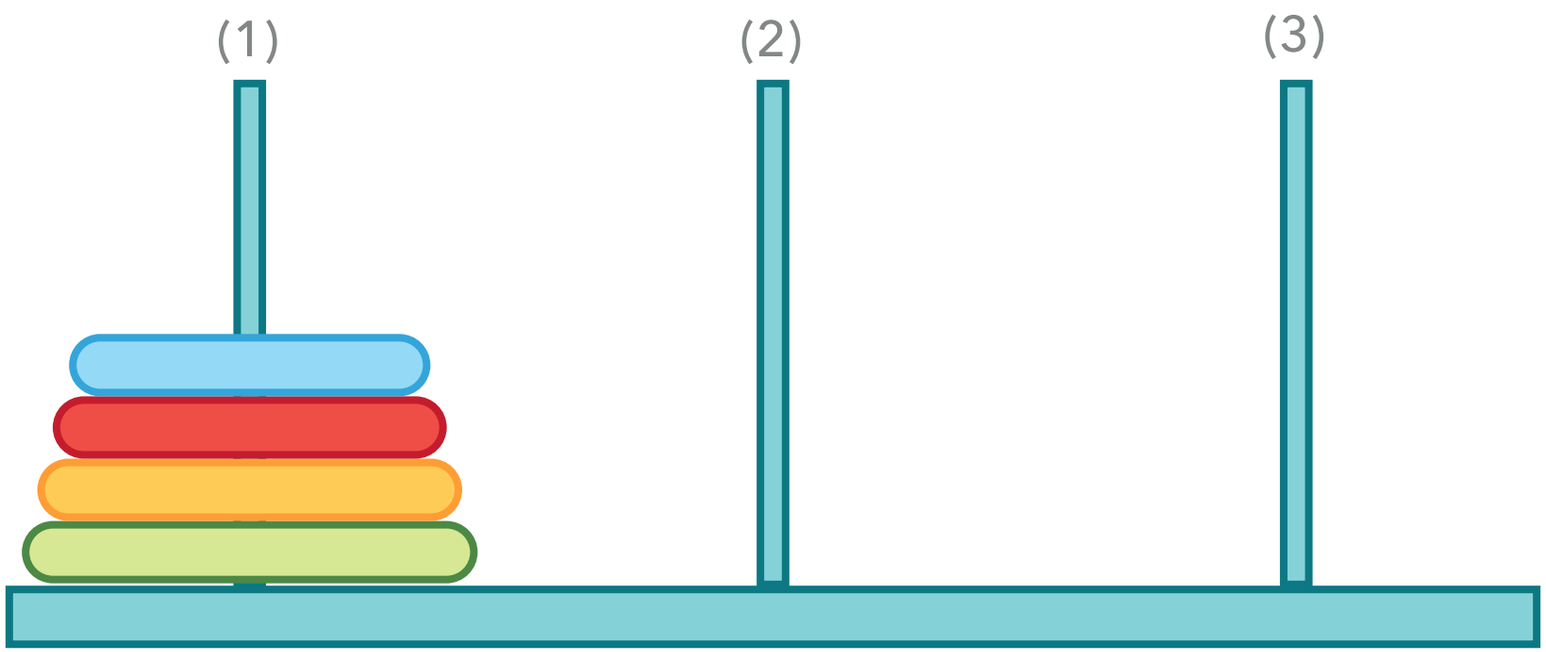
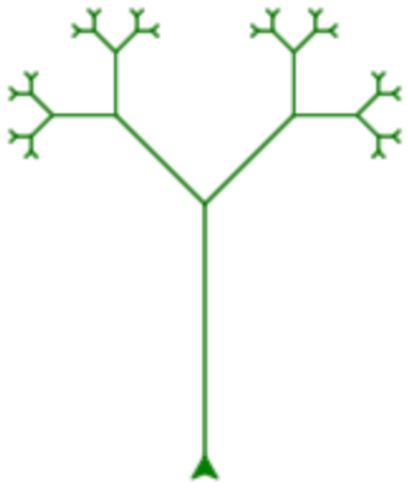


FIGURE RICORSIVE

Per illustrare la ricorsione proviamo ad usare Python per disegnare figure definite in modo ricorso

Per esempio questo “albero binario” generato in modo ricorsivo



LA "TARTARUGA"

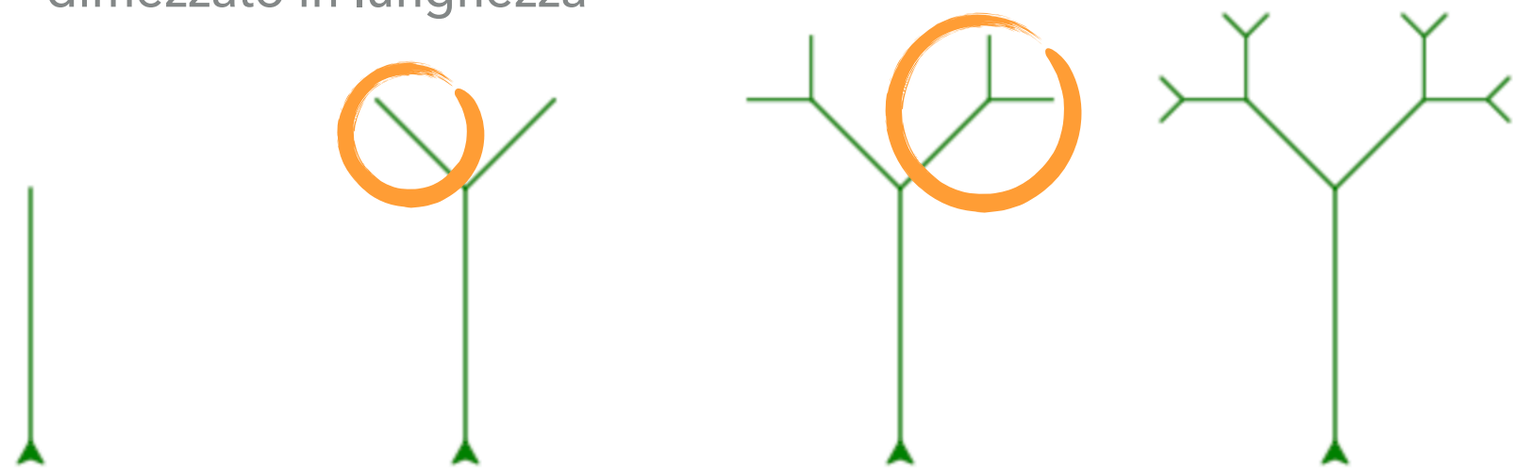
- In Python è possibile utilizzare una "tartaruga" per disegnare
- È uno strumento educativo che è ispirato ad un linguaggio pensato per insegnare la programmazione, LOGO
- È come avere un robot per disegnare al quale è possibile dare i comandi di muoversi avanti o indietro e di ruotare a destra o sinistra. Dove il robot passa traccia una linea

FIGURE RICORSIVE

Vediamo di individuare delle strutture in questo albero

È lo stesso dell'albero (1)
ma ruotato di 45° e
dimezzato in lunghezza

È lo stesso dell'albero (2)
ma ruotato di 45° e
dimezzato in lunghezza



(1) (2) (3) (4)

FIGURE RICORSIVE: DISEGNARE UN ALBERO

- Muoviti avanti di ℓ
- Ruota di 45° a sinistra
- Disegna un albero di lunghezza $\ell/2$
- Ruota di 90° a destra
- Disegna un albero di lunghezza $\ell/2$
- Ruota di 45° a sinistra (per tornare all'orientamento originale)
- Muoviti indietro di ℓ

QUALCHE ESERCIZIO

- Definire una funzione (non ricorsiva) che, dati n e ℓ, disegna il poligono regolare di n lati in cui ogni lato è lungo ℓ
- Definire una funzione (non ricorsiva) che, dato un array a e un numero x restituisce True se x è contenuto in a
- Riscrivere la precedente funzione in modo ricorsivo
- “Giocare” a modificare il disegno di alberi, che altri disegni possiamo fare? Provate a generare altre figure!