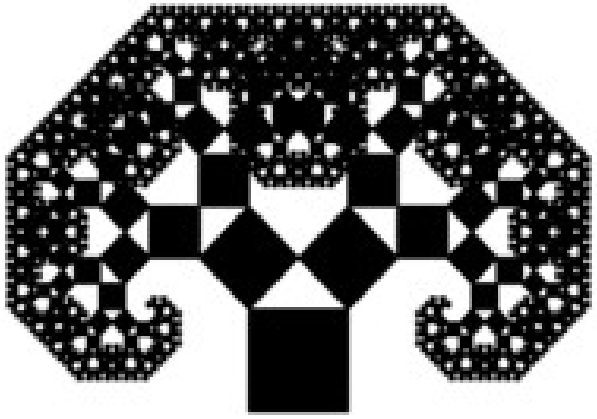
Recursivitat I/II

D.Tost

Recursivitat



Arbre de Pitàgores

«Una rosa és una rosa és una rosa» Gertrud Stein 1922

El matrimoni Arnolfini, Jan van Eyck, 1434

Recursivitat

-

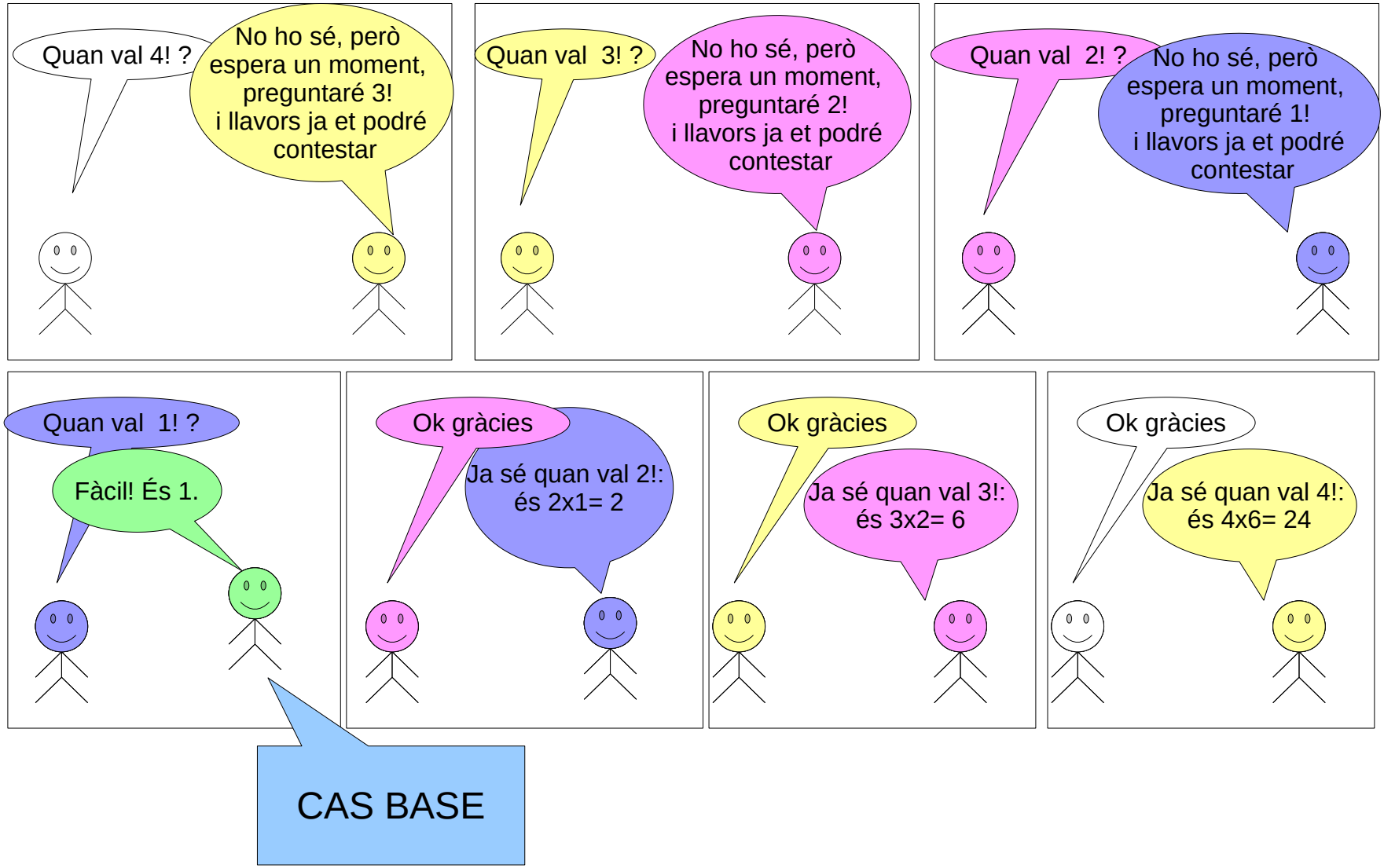
Un mètode en que la solució d‘un problema depèn de la solució del mateix problema aplicat a instàncies més simples del mateix problema

Recursivitat

- Metodologia:
  - Definir el cas que té una solució trivial que no depèn de la solució de subproblemes: **cas base**
  - Definir la **relació de recurrència**: la equació que defineix la solució del problema a partir de la solució de subproblemes més simples
  - Aplicar el patró següent:

```
def funció_recursiva (paràmetres)
    if cas_base:
        instruccions (sense crides a la funció)
    else:
        solució a partir del resultat d'una o més crides a funció_recursiva
```

Exemple factorial de 4



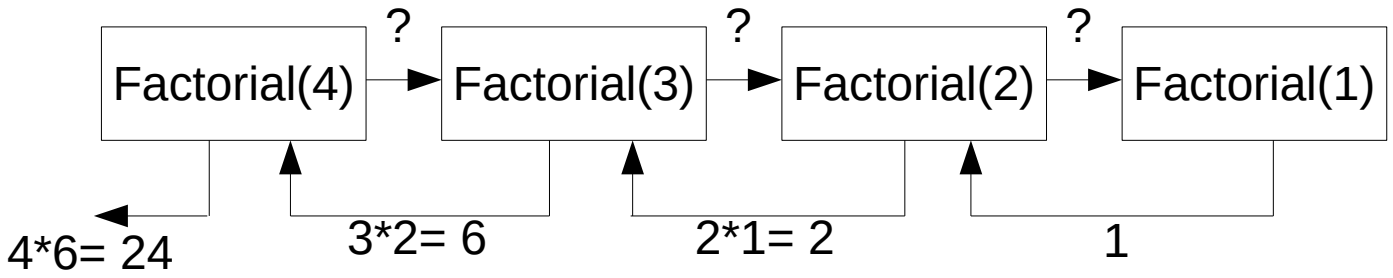
Exemple factorial de n.

- Relació de recurrència: n! = n *(n-1)!
- Cas base: 1! = 1
- Aplicació del patró:

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

```
def frec(paràmetres) :
        if cas_base :
            Solució trivial
        else:
            Crida/es recursiva/es a la frec
```

- Traça per n=4:



Límit de la recursivitat

```
>>> def no_acaba():
...     print ("una rosa és ...")
...     no_acaba()
...
>>> no_acaba()
una rosa és ...
una rosa és ...

[....]

una rosa és ...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in no_acaba
  File "<stdin>", line 3, in no_acaba
  File "<stdin>", line 3, in no_acaba
  [Previous line repeated 992 more times]
  File "<stdin>", line 2, in no_acaba
RecursionError: maximum recursion depth exceeded while calling a Python
object
una rosa és ...
```

Hi ha un límit a les crides recursives per sobre del qual python no suporta tenir més processos «en espera» de la resolució d'altres crides recursives.

```
>>> import sys
>>> sys.getrecursionlimit()
1000
```

Recursivitat terminal per la cua

- Quan la instrucció que acaba la funció és una crida a la funció sense cap operació («Tail recursive»). L'interès d'aquestes funcions és que en molts llenguatges de programació, les funcions recursives per la cua es poden optimitzar. No és cas de python, tot i que hi ha mòduls específics que ho intenten fer.

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

```
def factorialTR(n, v=1):
    if n == 0:
        return v
    return factorialTR(n-1, n*v)
```

`factorialTR` és «tail-recursive» perquè a la crida recursiva, retorna directament `factorialTR` sense cap operació. `factorial` en canvi no és perquè hi ha una operació a la crida recursiva.

Exemple 1 sèrie

Exemple: Dissenyeu la funció recursiva *terme* que donat un enter positiu *i*, retorna el terme t_i de la sèrie numèrica:

i = 1 -> $t_1 = 10$

i > 1 -> $t_i = 2t_{i-1} - 3$

```
def frec(paràmetres) :
        if cas_base :
            Solució trivial
        else:
            Crida/es recursiva/es a la frec
```

Cas base:

i = 1

terme(1) = 10

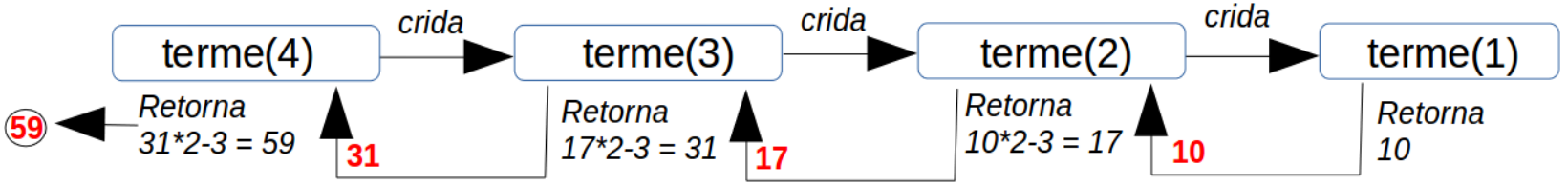
Cas recursiu:

i > 1

terme(i) = 2*terme(i-1) - 3

```
def terme(i) :
    if i == 1:
        return 10
    else :
        return 2*terme(i-1) - 3
```

Traça de `terme(4)`:



Exemple 2 sèrie

```
def frec(paràmetres) :
        if cas_base :
                return
        else:
                crida recursiva a la frec
```

Exemple: Dissenyeu la funció recursiva *texp* que donat un real x i un enter *i*, retorna el terme t_i del desenvolupament en sèrie de Taylor de la sèrie exponencial:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$

```
def texp(x, i) :
   if i == 0:
       return 1
   else:
       return x/i* texp(x, i-1)
```

Cas base:

`i = 0`

`texp(x, 0) = 1`

Cas recursiu:

`i ≥ 0`

$\texttt{texp(x, i)} = \frac{x^i}{i!}$

$= \frac{x \times x^{(i-1)}}{i \times (i-1)!}$

$= \frac{x}{i} \times \frac{x^{(i-1)}}{(i-1)!} = \frac{x}{i}$ `texp(x, i-1)`

Exemple 3 sèrie

Exemple: Dissenyeu la funció recursiva *exp* que donat un real x i un enter *n (n>0)*, retorna una aproximació de n+1 termes del desenvolupament en sèrie de Taylor de exp(x):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$

```
def frec(paràmetres) :
        if cas_base :
                return
        else:
                crida recursiva a la frec
```

Cas base:

$n = 1$

$exp(x, 1) = 1 + x$

Cas recursiu:

$n \geq 2$

$exp(x, n) = exp(x, n-1) + \frac{x^n}{n!}$

```
def exp(x, n) :
   if n == 1:
       return 1 + x
   else:
       return exp(x, n-1) + texp(x, n)
```

Exemple 3 sèrie

Exemple: Dissenyeu la funció recursiva *exp* que donat un real `x` i un enter *`n (n>0)`*, retorna una aproximació de n termes del desenvolupament en sèrie de Taylor de exp(x):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$

```
def frec(paràmetres) :
        if cas_base :
                return
        else:
                crida recursiva a la frec
```

Inconvenient de la solució anterior: quan calculem `texp(x, i)` per sumar-lo a `exp(x, i-1)`, tornem a calcular tots els termes anteriors. És molt poc eficient! Una alternativa seria que la funció retornés també el terme anterior a més de la suma dels termes anterior. Així, no recalcularíem tot cada vegada. El problema llavors és que no volem que `exp` retorni 2 valors sinó només 1.
Solució: fem una funció que retorna només la suma i crida la funció recursiva que retorna el terme i la suma.

```
def exp2(x, n):
    return exp2_rec(x, n)[1]
```

```
def exp2_rec(x, n):
    if n == 1:
        return  x, 1 + x
    else:
        terme, suma = exp2_rec(x, n-1)
        nou_terme = x/n*terme
        return nou_terme, nou_terme + suma
```

Exemple 4 sèrie

```
def frec(paràmetres) :
        if cas_base :
                return
        else:
                crida recursiva a la frec
```

Exemple: Dissenyeu la funció recursiva *tcos* que donat un angle *x* en radiants i un enter parell *i*, retorna el terme t_i de la sèrie cosinus:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

```
def tcos(x, i) :
    if i == 0:
        return 1
    else :
        t = tcos(x, i-2)
        t = -t*(x**2)/((i-1)*i)
        return t
```

Cas base:

`i = 0`
`tcos(x, 0) = 1`

Cas recursiu:

`i ≥ 0` (i `i%2 == 0`)

$tcos(x, i) = \pm x^i/i!$

$= \pm (x^2 * x^{(i-2)})/((i-2)!*(i-1)*(i))$

$= - tcos(x, i-2) * x^2/((i-1)*i))$

Exemple 5 sèrie

Exemple: Dissenyeu la funció recursiva `S(n, i)` que donat un enter `n` i un enter `i` estrictament positiu retorna el `i`-èssim element de la sèrie:

$$x_1 = n$$

$$x_{i+1} = \begin{cases} x_i/2 & \text{si } x_i \text{ es parell,} \\ 3x_i + 1 & \text{si } x_i \text{ es senar} \end{cases} \quad \text{per } i > 0$$

```
def frec(paràmetres) :
        if cas_base :
                return
        else:
                crida recursiva a la frec
```

Per calcular x_i cal conèixer x_{i-1} i per calcular x_{i-1} cal conèixer x_{i-2} i així successivament fins a arribar a i = 1, valor que es coneix (n) i a partir del qual es pot anar calculant tots els valors fins a x_i.

```
def S(n, i) :
    if i == 1:
        return n
    else :
        x = S(n, i-1)
        if x%2 == 0:
            return x/2
        else:
            return 3*x +1
```

Cas base:

i = 1 → valor retornat n

Cas recursiu:

i > 1 → valor retornat serà S(n, i-1)/2 o 3*S(n, i-1) +1 dependent de si S(n, i-1) és parell o senar

Exemple 6 sèrie

Exemple: Dissenyeu la funció recursiva `S(n)` que donat un enter `n` retorna el `n`-èssim element de la sèrie. Alerta ! És diferent de l'anterior!!!

$$x_1 = n$$

$$x_{i+1} = \begin{cases} x_i/2 & \text{si } x_i \text{ es parell,} \\ 3x_i + 1 & \text{si } x_i \text{ es senar} \end{cases} \quad \text{per } i > 0$$

Calen igualment dos paràmetres: *n* (per calcular el cas base) i *i* per identificar el terme i-èssim. Segons l'enunciat però, S només té un paràmetre. La solució és que S(n) cridi una funció auxiliar Saux(n, i)
que tindrà els dos paràmetres necessaris i serà recursiva.

```
def S(n):
    return Saux(n , n)


def Saux(n, i) :
   if i == 1:
       return n
   else :
       x = S(n, i-1)
       if x%2 == 0:
          return x/2
       else:
          return 3*x +1
```

Exemple 7 llista i recorregut

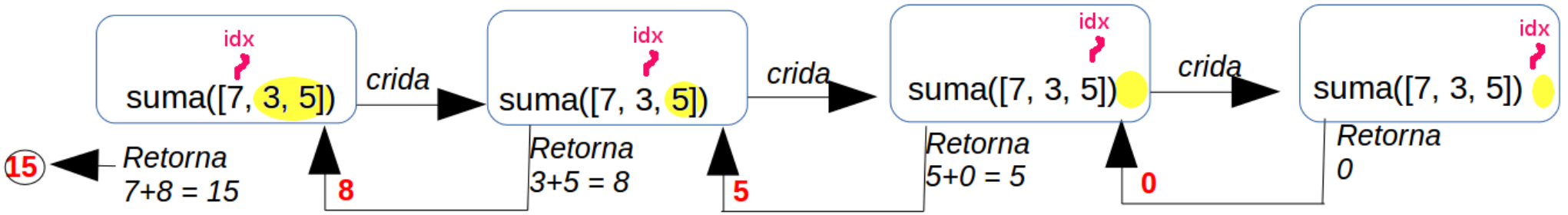
Exemple: Dissenyeu la funció recursiva *suma* que donada una llista retorni la suma dels seus elements

Donada la suma dels elements de la resta de la llista a partir d'una posició idx (llista[idx+1]), la suma dels elements de llista[idx:] és llista[idx] + suma(llista[idx+1:].

Imaginem que tenem una variable idx que comença amb valor 0 i va avançant sobre cadascun del elements de la llista.

Cas base: idx == len(llista) → 0

Cas recursiu: → suma(llista, idx+1) + llista[idx]



Exemple 7 llista i recorregut

Versió 1

Exemple: Dissenyeu la funció recursiva *suma* que donada una llista retorni la suma dels seus elements

Com en l'exercici anterior, cridem una funció auxiliar que tindrà dos paràmetres: la llista i l'índex. La llista es passa com a paràmetre en crides successives de la funció. L'enter *idx* serveix per saber quin tros de la llista cal visitar (de idx en endavant).

```
def suma(l):
    return suma_aux(l, 0)

def suma_aux(l, idx) :
    if idx == len(l):
        return 0
    else :
        return l[idx] + suma_aux(l, idx+1)
```

```
>>> sum([7, 3, 5)
18
```

Exemple 7 llista i recorregut

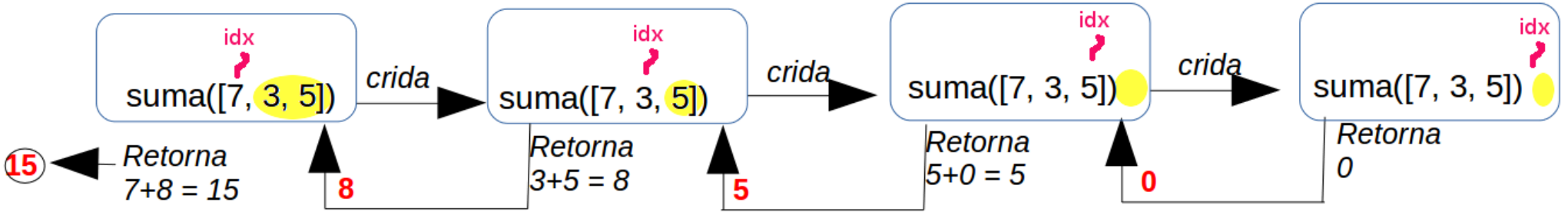
Versió 2

Exemple: Dissenyeu la funció recursiva *suma* que donada una llista retorni la suma dels seus elements

Si volem evitar la funció auxiliar, podem afegir el paràmetre idx amb un valor de defecte (0).

```
def suma(l, idx=0):
    if idx == len(l):
        return 0
    else :
        return l[idx] + suma(l, idx+1)
```

```
>>> sum([7, 3, 5)
18
```



Exemple 7 llista i recorregut

Versió 3

Exemple: Dissenyeu la funció recursiva *suma* que donada una llista retorni la suma dels seus elements

Cas base:

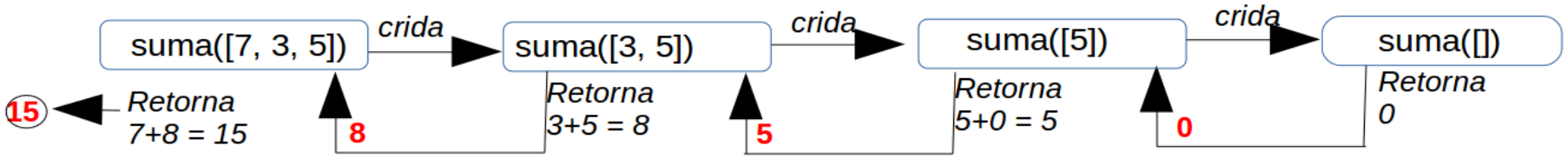
len(l) == 0 → suma(l) = 0

Cas recursiu:

len(l) > 0 → suma(l) = l[0] +suma(l[1:])

```
def suma(l) :
   if len(l) == 0:
       return 0
   else :
       return l[0] + suma(l[1:])
```

Es crida de forma recursiva la funció amb subllistes resultants d'escapçar el primer element. **L'inconvenient d'aquesta solució és que crea subllistes i no es té informació del índex de l'element actual** (caldria passar-lo específicament), el que pot ser una limitació per alguns problemes específics.

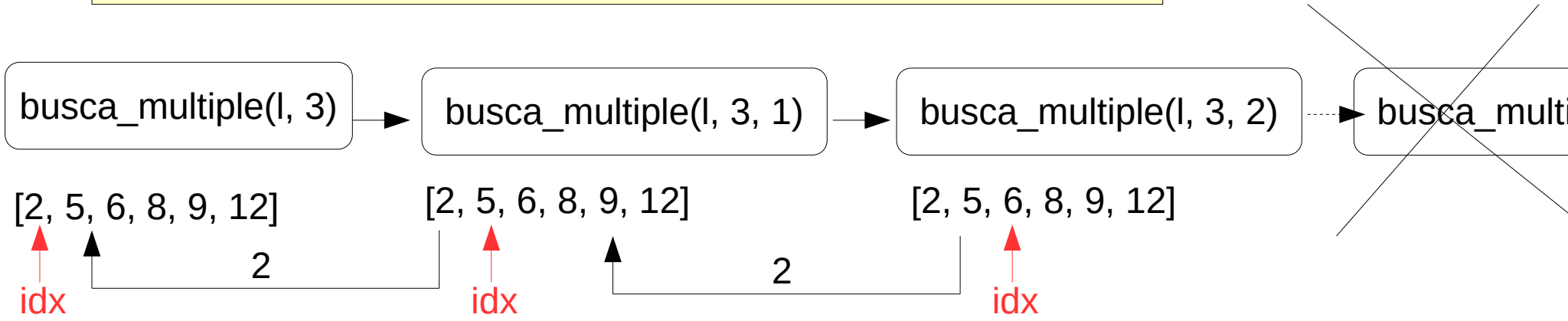


Exemple 8: llistes i cerca

Exemple: Dissenyeu la funció recursiva *busca_multiple* que donada una llista *l* i un enter *m* retorni l'índex del primer element de la llista que és múltiple de *m*. Si no en hi ha, cal retornar -1

```
def busca_multiple(l, m, idx = 0) :
    if idx == len(l):
        return -1
    else :
        if l[idx]%m == 0:
            return idx
        else :
            return  busca_multiple(l, m, idx+1)
```

A la que troba el múltiple el retorna a la crida anterior



Més la setmana vinent