Recursivitat II/II

D.Tost

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 9 llista i cerca

Exemple: Dissenyeu la funció recursiva *capicua* que donat un string *s* indiqui si és capicua (palíndrom). Es espais en blanc no compten ni tampoc les majúscules, e.g: «Un aviador roda i va nu» és palíndrom.

```
def capicua(s):
    s2 = s.replace(' ', '').lower()
    return capicua_rec(s2, 0)

def capicua_rec(s, idx):
    if idx >=len(s)/2 :
        return True
    else:
        if s[idx] != s[-1-idx]:
            return False
        return capicua_rec(s, idx+1)
```

Alerta! No confondre la condició de cerca i el cas base

0 1 2
abccba len(s)/2=6/2=3

0 1 2
abcdcba len(s)/2=7/2=3

Exemple 10 Llista de persones

Dissenyeu la funció *busca_menor* que donada una llista d'instàncies de la classe `Persona`, retorni True si hi ha una persona de la llista que sigui menor d'edat i retorni False en cas contrari

```
classe Persona(nom, edat)

    Atributs:

        nom: nom de la persona (str)

        edat: edat de la persona (int)

    Suporta la funció str
```

```
def busca_menor(l, idx = 0):
    if idx == len(l):
        return False
    else :
        if l[idx].edat <18:
            return True
        else :
            return busca_menor(l, idx+1)
```

La persona en curs és menor, no cal seguir buscant

La persona en curs no és menor, n'hi haurà una si n'hi ha una a la resta de la llista

Hem recorregut tota llista i no n'hem trobat cap

```
>>> p = busca_menor([Persona('A', 24), Persona('B', 23)])
False
>>> p = busca_menor([Persona('A', 24), Persona('B', 2)])
True
```

Exemple 11 Llista de persones

Dissenyeu la funció *busca_persona_menor* que donada una llista d'instàncies de la classe Persona, retorni la primera persona de la llista que sigui menor d'edat i una persona de nom '' i edat 0 en cas contrari.

```
classe Persona(nom, edat)
     Atributs:
         nom: nom de la persona (str)
         edat: edat de la persona (int)
     Suporta la funció str
```

```
def busca_persona_menor(l, idx = 0):
  if idx == len(l):
     return Persona('', 0)
  else :
     if l[idx].edat <18:
        return l[idx]
     else :
        return busca_persona_menor(l, idx+1)
```

```
>>> llista = [Persona('A', 24), Persona('B', 23), Persona('C', 59),
              Persona('D', 5), Persona('E', 3)]
>>> menor = busca_persona_menor(llista)
>>> str(menor)
'Persona D de 5 anys'
```

Exemple 12 Llista de persones

Dissenyeu la funció *mes_gran* que donada una llista d'instàncies de la classe Persona, retorni el nom de la persona més gran. Suposeu que la llista no és buida.

```
classe Persona(nom, edat)
     Atributs:
         nom: nom de la persona (str)
         edat: edat de la persona (int)
     Suporta la funció str
```

```
def mes_gran(l) :
    p = persona_mesgran(l)
    return p.nom

def persona_mesgran(l, idx = 0):
     if idx == len(l) -1 :
         return l[idx]
     else :
         p = persona_mesgran(l, idx+1)
         if p.edat > l[idx].edat :
            return p
         else :
            return l[idx]
```

Comparem edats, però hem de retornar el nom. Per això, fem fem dues funcions: la que ens han encarregat `(mesgran)` i la recursiva `(persona_mesgran)` que retorna una persona, no una edat.

Exemple 13 Llista de jugadores

La classe jugadora representa les persones que juguen en un torneig de 10 partides d'un joc. Està especificada a continuació. Dissenyeu la funció `guanyadora` que donada una llista d'instàncies de la classe `Jugadora`, retorni el nom de la persona jugadora amb més punts. Suposeu que la llista no és buida.

```
classe Jugadora(nom)
     Retorna una Jugadora amb 0 punts pels 10 partits del torneig

     Atributs:
         nom: nom de la persona (str)
     Operacions:
```

g[i]	retorna els punts de la partida i-èssima $0 \leq i < 10$
g[i] = punts	assigna els punts a la partida i-èssima $0 \leq i < 10$

```
    Mètodes:
        total_punts(): retorna la puntuació total de la jugadora
    Suporta la funció str
```

Exemple 13 Llista de jugadores

```
def guanyadora(llista):
    idx = guanyadora_rec(llista, 0)
    return llista[idx].nom

def guanyadora_rec(llista, idx):
    if idx == len(llista)-1:
        return idx
    else:
        idx_guanya = guanyadora_rec(llista, idx +1)
        if llista[idx].total_punts() > llista[idx_guanya].total_punts():
            return idx
        else:
            return idx_guanya
```

La funció `guanyadora_rec` retorna l'índex de la persona guanyadora. A partir de l'índex, la funció `guanyadora`, retorna el nom. Observeu que seria trivial que retornés la jugadora o els seus punts.

Per calcular el punts, hem d'invocar el mètode `punts_total()`.

Exemple 14 Crear llistes

Dissenyeu la funció recursiva `crea_llista_m` que crea la llista dels `n` primers múltiples de `m`

Cas base:

n == 0 → []

Cas recursiu

n >0 → crea_llista_m(n-1, m) + [n*m]

Múltiples per i variant de 1 a n

```
def crea_llista_m(n, m) :
    if  n==0:
        return []
    else :
        llista = crea_llista_m(n-1, m)
        return [n*m] + llista
```

Els volem de més gran a més petit

Exemple 14 Crear llistes

- Dissenyeu una funció recursiva que crea la llista del `n` primers múltiples de `m`.
- Segona versió: Volem aprofitar que amb el múltiple anterior, sumant-li `m`, ja tenim el nou múltiple (serà més eficient)
- La generem de primer múltiple a darrer múltiple

```
def crea_llista_m(n, m) :
    if n == 1:
        return [m]
    else :
        llista = crea_llista_m(n-1, m)
        llista.append(llista[-1] + m)
        return llista
```

El darrer element de la llista és el darrer múltiple: només cal sumar-li `m` i afegir-lo a la llista

Exemple 15 Classes contenidores

La classe Grup representa un grup de persones

Dissenyeu el mètode `mitjana_edat`, que retorna la mitjana d'edat de les persones del grup.

```
>>> from exemple import Persona, Grup
>>> p = Persona('A', 32)
>>> p2 = Persona('B', 81)
>>> g = Grup('G1')
>>> g.afeg(p2)
>>> g.afeg(p)
>>> g.mitjana_edat()
56.5
```

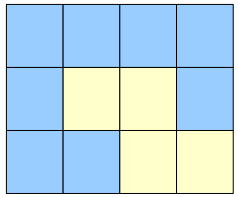
`mitjana_edat` no és recursiu però crida al mètode auxiliar `suma_edat` que sí ho és.

```
class Grup:
    def __init__(self, nom):
        self.nom_grup = nom
        self.__persones = []

    def __getitem__(self, i):
        return self.__persones[i]

    def afegir(self, p):
        self.__persones.append(p)

    def __len__(self):
        return len(self.__persones)

    def suma_edat(self, idx = 0):
        if idx == len(self):
            return 0
        else:
            return self.suma_edat(idx+1) + self[idx].edat

    def mitjana_edat(self):
        if len(self) > 0:
            return self.suma_edat(0)/len(self)
        else:
            return 0
```

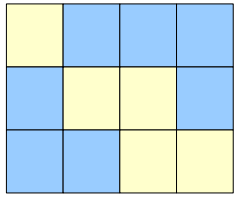
Exemple 16 Classes contenidores

La classe `Teatre` representa els seients d'un teatre organitzats en una retícula de n files i m columnes.

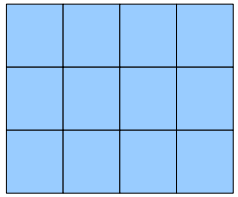
Dissenyeu el mètode *`busca_seient`* de la classe `Teatre`, que retorna la posició del primer seient lliure, un tuple (fila, columna) i (-1, -1) si no n'hi ha.



(0, 0) → (0, 1) → (0, 2) → (0, 3) → (1, 0) → **(1, 1)**



(0, 0)



(0, 0) → (0, 1) → (0, 2) → (0, 3) → (1, 0) → (1, 1) → (1, 2) → (1, 3) → (2, 0) → (2, 1) → (2, 2) → (2, 3) → **(3, 0)**

```
class Teatre :
    def __init__(self, n, m) :
        self.files = n
        self.columnes = m
        self.__seients = []
        for i in range(n) :
            self.__seients.append(['buit']*m)

    def __getitem__(self, pos) :
        return self.__seients[pos[0]][pos[1]]

    def __setitem__(self, pos, nom) :
        self.__seients[pos[0]][pos[1]] = nom

    def busca_seient(self, i = 0, j = 0):
        if i ==  self.files:
            return (-1, -1)
        else:
            if self[i, j] == 'buit':
                return (i, j)
            else:
                j = j + 1
                if j == self.columnes:
                    i = i + 1
                    j = 0
                return self.busca_seient(i, j)
```

Exemple 17 Fractals

Hem de crear de forma recursiva una llista de quadrats. Per cada quadrat se'n generen 2 que a la vegada generen 2 nous quadrats i així successivament.

Donat un quadrat, com calcular els seus dos descendents immediats?

$$mida(q1) = \frac{mida(q)}{\sqrt{2}}$$

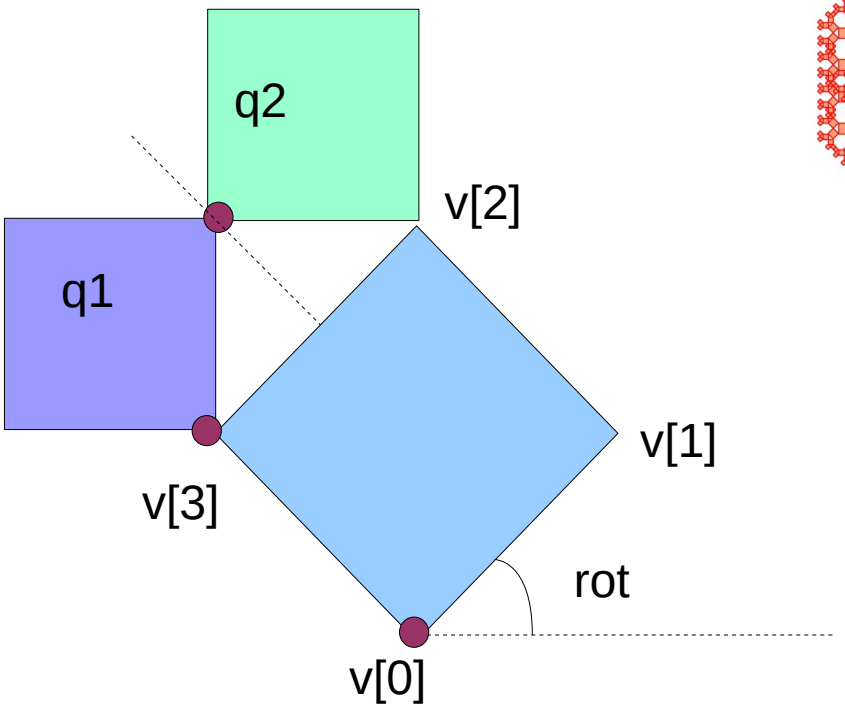
$$mida(q2) = mida(q1)$$

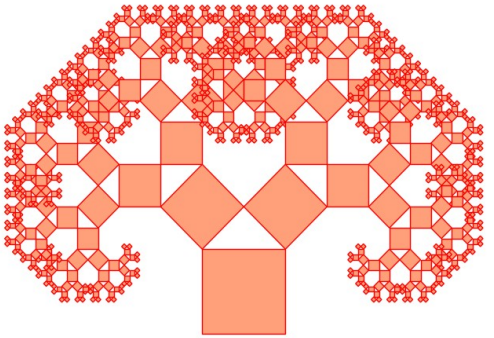
$$rot(q1) = rot(q) + \frac{pi}{4}$$

$$rot(q2) = rot(q) - \frac{pi}{4}$$

$$v[0](q1) = v[3](q[0])$$

$$v[0](q2) = v[1](q1)$$





Exemple 17 Fractals

Dissenyeu la funció `crear_arbre_rec(mida, pos, n)` que, donada la fondària de l'arbre, retorna la llista de quadrats que formen l'arbre. Els quadrats són instàncies de la classe `Quadrat` especificada a continuació.

```
class Quadrat(mida, pos, rot)
   Representa un quadrat amb el vèrtex inferior esquerra a pos
   i girat de rot respecte a l'eix x
   Atributs:
    rot: l'angle de gir del quadrat respecte a l'eix x (en radiants)
    mida: la mida de l'aresta del quadrat
       vertexs: una llista de dues subllistes amb les coordenades x i y
       dels 4 vèrtexs del  quadrat

   Suporta l'operador q[i] amb i =0...3 que retorna el vèrtex i-èssim
   del quadrat (un tuple(x, y))
```

La dificultat geomètrica del problema està resolta en la classe `Quadrat` on es calculen les coordenades del 4 vèrtexs a partir de pos i rot.

Es proporciona la funció `dibuixa_quadrats(llista)` per dibuixar la llista de quadrats

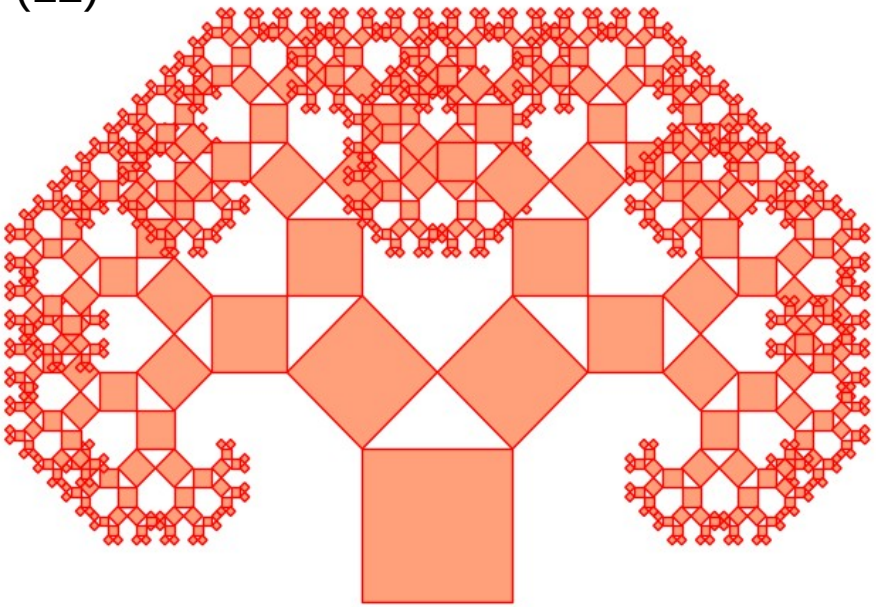
Exemple 17 Fractals

Cas base: n = 0 → llista buida
Cas recursiu:

Donat un quadrat q

- calcular q1 i q2
- obtenir la llista de descendents de q1 (L1)
- obtenir la llista de descendents de q2 (L2)

→ [q] +[L1]+ [L2]



```
def crear_arbre(mida, pos, n):
    return crear_arbre_rec(Quadrat(mida, pos, 0), n)

def crear_arbre_rec(base, n):
    if n == 0:
        return []
    else:
        nova_mida = base.mida/arr2
        q1 = Quadrat(nova_mida, base[3], base.rot + pi/4)
        llista1 = crear_arbre_rec(q1, n-1)
        q2 = Quadrat(nova_mida, q1[1], base.rot - pi/4)
        llista2 = crear_arbre_rec(q2, n-1)
        return [base] + llista1 + llista2
```