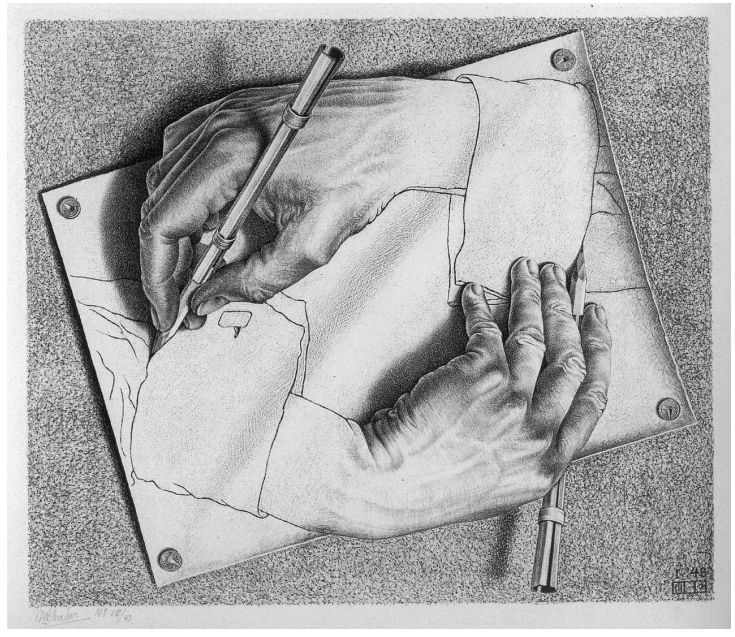
INFO-H-100 - Informatique

Séance d'exercices 16

La récursivité



Université libre de Bruxelles – Faculté des Sciences Appliquées – 2013-2014

Quelques exemples

C'est l'histoire d'un enfant...
qui n'arrivait pas à dormir et à qui sa mère raconta l'histoire d'un lapin
qui n'arrivait pas à dormir et à qui sa mère raconta l'histoire d'une chèvre
qui n'arrivait pas à dormir et à qui sa mère raconta l'histoire d'un cochon
qui n'arrivait pas à dormir et à qui sa mère raconta l'histoire d'une vache
qui n'arrivait pas à dormir et à qui sa mère raconta l'histoire d'un cheval

⋮

qui s'endormit

⋮

et qui alors s'endormit
et qui alors s'endormit
et qui alors s'endormit
et qui alors s'endormit
qui alors s'endormit.

Quelques exemples

Récursivité

La récursivité se base sur le principe de récurrence.

Principe de récurrence

Soit P_n, une proposition relative à l'entier n et n_0 un entier.

- si la proposition P_{n_0} est vraie (initialisation)
- et si pour tout $k > n_0$, la proposition P_k vraie implique que la proposition P_{k+1} soit vraie (hérédité),

alors la proposition P_n est vraie pour tout entier $n \geq n_0$.

Fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même.

Exemple : factorielle

- initialisation : $0! = 1$
- hérédité : $n! = n \times (n-1)!$

En Python :

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

Exécution : http ://www.pythontutor.com/visualize.html

Pratiquement

Lorsque vous écrivez une fonction récursive, assurez-vous :

1. que chaque appel récursif réduise la taille du problème (jusqu'à atteindre le cas de base)
2. que le cas de base (initialisation) soit bien géré

Si une récursion n'atteint jamais le cas de base, le programme ne se termine (théoriquement) jamais : c'est une récursion infinie.

```
def recurse():
    recurse()
```

Attention !

Une fonction récursive est **plus lente** et utilise **plus de mémoire** qu'un équivalent itératif (basé sur une boucle).

*"En règle générale, utilisez la récursivité si et seulement si, à chaque itération, la tâche se divise en **deux ou plus** tâches similaires ; sinon, préférez une implémentation itérative."*

Autre exemple de *mauvaise* utilisation : **la factorielle**

```
def factorial_recursive( n ):
    if n < 1:
        res = 1
    else:
        res = n * factorial_recursive( n-1 )
    return res
```

```
def factorial_iterative( n ):
    res = 1
    for i in range(2,n+1):
        res *= i
    return res
```

$$\frac{\text{temps récursif}}{\text{temps itératif}} \sim 2.5$$

Testez le vous-même avec le script `timecomp.py`

Exemple de *mauvaise* utilisation : **le projet Boggle**

```
def play_again():
    answer = input("Voulez-vous rejouer (o/n) ?")
    while answer != 'o' and answer != 'n':
        answer = input("Voulez-vous rejouer (o/n) ?")
    if answer == 'o':
        play_boggle()
    else:
        print("Au revoir !")

def play_boggle():
    print("Le jeu a lieu...")
    print("Le jeu se termine...")
    play_again()
```

Exemple de *bonne* utilisation : **les tours de Hanoï**

Exemple de *bonne* utilisation : **les tours de Hanoï**

```
def recursive_hanoi(n, source, helper, target):
    if n > 0:
        # move tower of size n - 1 to helper:
        recursive_hanoi(n - 1, source, target, helper)
        # move disk from source peg to target peg
        if source:
            disk = source.pop()
            target.append(disk)
        # move tower of size n-1 from helper to target
        recursive_hanoi(n - 1, helper, source, target)


def hanoi(n):
    source = list(range(n,0,-1))
    helper = []
    target = []
    recursive_hanoi ( n, source, helper, target )
```

Testez le vous-même avec le script `hanoi.py`

Autre exemple de *bonne* utilisation : **Sudoku**

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Autre exemple de *bonne* utilisation : **Sudoku**

```
def recursive_solve_sudoku( grid, row, col ):
    original_value = grid[row][col]
    possible_values = valid_values( grid, row, col )
    res = False
    if len(possible_values) > 0:
       if (row,col) == (8,8):  # last position?
          grid[row][col] = possible_values[0]
          res = True
       else:
          (nxtrow,nxtcol) = next_grid_pos( row, col )
          i = 0
          while not res and i < len(possible_values):
             grid[row][col] = possible_values[i]
             res = recursive_solve_sudoku( grid, nxtrow, nxtcol )
             i += 1
          if not res:
             grid[row][col] = original_value
   return res

def solve_sudoku( grid ):
   return recursive_solve_sudoku( grid, 0, 0 )
```

Testez le vous-même avec le script `sudoku.py`

Python

Le niveau maximum de récursion dans Python est par défaut fixé à 1000.

```
>>> factorial(1000)
RuntimeError: maximum recursion depth exceeded in comparison
```

Exercices

Cette séance étant destinée à vous familiariser avec les bases de la récursivité, les exercices proposés correspondent à des cas simples pour lesquels une version iterative serait généralement plus efficace.

Exercices complémentaires (facultatifs) :

- Pour quelques exercices, programmez la variante itérative et comparez-la en termes de lisibilité à la version récursive.
- Reprenez le code de l'arbre tracé avec Turtle à l'exercice 3 de la séance 13 et revoyez-le à la lumière de la récursivité. Adaptez ensuite le code pour tracer des flocons de Koch. Quelle est la différence d'implémentation majeure ?
- Programmez vous-même l'algorithme de résolution des tours de Hanoï.