Informatique 3 — Découverte de la récursivité

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

– Exemple fondamental : la factorielle –

Voici une méthode bien connue :

```
def factoI(n):
    r=1
    for i in range(1,n+1):
          r=r*i
    return r
print("la factorielle de ",5," vaut ",factoI(5))
```

La voici sous une autre forme :

```
def factoR(n):
    if n<=1: return 1
    else: return n*factoR(n-1)
print("la factorielle de ",5," vaut ",factoR(5))
```

Discutons, comparons, décortiquons ces deux implémentations. Nombre d'opérations ? Place mémoire utilisée ?

Un peu comme pour définir une suite par récurrence en Maths, il faut :

1. Un (ou plusieurs) cas de base, dans laquelle l'algorithme ne s'appelle pas lui-même. Sinon l'algorithme ne peut pas terminer !
2. Si on ne se situe pas dans un cas de base (on parle de « cas inductif »), alors l'algorithme fait appel à lui-même (appel récursif).

Chaque appel récursif doit en principe se « rapprocher » d'un cas de base, de façon à permettre la fin du programme.

L'appel des fonctions est géré à l'aide d'une pile d'exécution qui conserve les contextes d'appel : noms de variables locales, leur contenu... À chaque appel de fonction, on empile le contexte ; à chaque sortie de fonction, on dépile !

Tout algorithme possède une version itérative ou une version récursive. Parfois l'une des deux formes est plus appropriée (efficacité ou simplicité) que l'autre.

1. La récursivité a essentiellement deux avantages :
   - Décomposer une action répétitive en sous-actions identiques de petite taille.
   - Explorer un ensemble de possibilités.
2. La récursivité a essentiellement un inconvénient :
   - Mal gérée, elle peut mener à des complexités exponentielles !

Exercice 1 : qui suis-je ?

On définit une méthode `python` comme suit :

```
def r(x,deb,fin,L):
    if deb>fin: return "fini"
    t=(deb+fin)//2
    if x==L[t]: return t
    if x<L[t]: return r(x,deb,t-1,L)
    else: return r(x,t+1,fin,L)
L=[2,4,6,7,9,13,26]
print("liste initiale : ",L)
print("résultat : ",r(13,0,6,L))
```

Décrire sur papier les étapes lors de l'appel ci-dessus. Quel est le nom de l'algorithme utilisé ici ?

Exercice 2 : qui suis-je ?

On définit une méthode `python` comme suit :

```
def ed(L,M):
    if L==[]: return M
    a=L.pop(0)
    if a not in M: M.append(a)
    return ed(L,M)
L=[2,3,2,6,8,9,9,10,9,3,6,7,8,8,9]
print("liste initiale : ",L," et résultat : ",ed(L,[]))
```

Décrire sur papier les étapes lors de l'appel ci-dessus. Que fait cet algorithme ?
Écrire une méthode itérative `ed2` ayant le même effet que `ed`.

Exercice 3 : qui suis-je ?

On définit une méthode `python` comme suit :

```
def pp(L):
    m=L[0]
    for i in range(1,len(L)-1):
          if L[i]<m:  m=L[i]
    return m
L=[24,45,2,3,2,6,8,9,9,10,9,3,6,7,8,8,9]
print("la liste :",L," et le résultat de méthode pp :",pp(L))
```

Décrire sur papier les étapes lors de l'appel ci-dessus. Que fait cet algorithme ?
Écrire une méthode récursive `pp2` ayant le même effet que `pp`.

Exercice 4 : calcul du PGCD par l'algorithme d'Euclide

À l'aide des propriétés :

- pour tous entiers a et b, on a $\text{pgcd}(a,b) = \text{pgcd}(a-b,b)$;
- pour tous entiers a et b, on a $\text{pgcd}(a,b) = \text{pgcd}(b,a)$;
- pour tout entier a, on a $\text{pgcd}(a,0) = a$;

écrire une fonction récursive `pgcd` retournant le PGCD de deux entiers.

Exercice 5 : combinaisons

À l'aide des trois propriétés

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}, \quad \binom{n}{n} = 1 \quad \text{et} \quad \binom{n}{1} = n,$$

écrire une fonction récursive `combi` retournant le nombre $\binom{n}{k}$ de combinaisons de k éléments parmi n.

Exercice 6 : algorithme de Héron d'Alexandrie (1e siècle ap. J.-C.)

Pour déterminer la racine carrée du nombre A, on choisit un nombre x_0 (autant que possible proche de $\sqrt{A}$, par exemple la partie entière de $\sqrt{A}$). Puis on construit une suite définie par récurrence par $x_{n+1} = \dfrac{x_n + \frac{A}{x_n}}{2}$. La suite ainsi obtenue est une suite positive, décroissante à partir du second terme, et convergeant vers $\sqrt{A}$.

Écrire deux méthodes `HeronI` et `HeronR`, respectivement itérative et récursive, calculant les n premiers termes de la suite de Héron de la racine d'un entier A à partir d'un réel x_0 arbitrairement fixé. On affichera les x_n pour $n \leq 5$.
Remarque 1 : la convergence est quadratique, c'est-à-dire que l'écart entre un terme et la limite $\sqrt{A}$ évolue comme le carré de l'écart précédent : $x_{n+1} - \sqrt{A} = \left(x_n - \sqrt{A}\right)^2 \dfrac{1}{2x_n}$. Ainsi le nombre de décimales exactes double à chaque itération !
Remarque 2 : bien qu'attribué à Héron d'Alexandrie, cet algorithme était déjà connu à Babylone vers 600 av. J.-C.

Exercice 7 : nombres de chiffres

On rappelle que le quotient de la division euclidienne d'un entier n par 10 donne le nombre de dizaines de cet entier. Le quotient de la division euclidienne de $n = 2018$ par 10 est par exemple 201.

Écrire une fonction `NbChiffres(n)` récursive prenant en paramètre un entier naturel n (écrit en décimal) et retournant le nombre de chiffres de cet entier n en base 10.

Exercice 8 : suite de Fibonacci

On appelle suite de Fibonacci la suite d'entiers définie par :

$$u_0 = u_1 = 1 \quad \text{et} \quad \forall n \geq 2,\ u_n = u_{n-1} + u_{n-2}.$$

1. Écrire deux procédures `FibonacciR(n)` et `FibonacciI(n)`, respectivement récursives et itératives, calculant le n-ième terme de la suite de Fibonacci.
2. Comparer les deux procédures précédentes en calculant u_n pour $n \in \{20, 25, 30, 35, 40\}$. On pourra importer le module `time` et utiliser la fonction `time.time()` de la façon suivante :

```
⟨ implémentation des procédures ⟩
debut = time.time()
⟨ calculs dont on veut mesurer l'efficacité ⟩
print("temps de calcul :",time.time()-debut)
```

Exercice 9 : valeur approchée de π

Écrire une fonction récursive `wallis(n)` qui renvoie une valeur approchée de π grâce à la formule

$$\pi = \lim_{n \to +\infty} 2 \prod_{i=1}^{n} \frac{4i^2}{4i^2 - 1}.$$

Ici n indique le nombre de termes du produit.

Exercice 10 : chiffres romains

Écrire une fonction récursive `romain2decimal(N)` qui renvoie la traduction en chiffres arabes du nombre `N` écrit en chiffres romains. On rappelle que

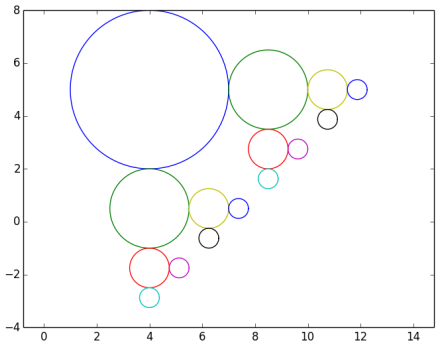
$$M = 1000, \quad D = 500, \quad C = 100, \quad L = 50, \quad X = 10, \quad V = 5, \quad I = 1.$$

Par exemple $MMXVIII$ signifie 2018.

La tester sur les nombres $MMXIV$ à $MMXX$. Ne pas oublier les cas particuliers, tels $IV = 4$!

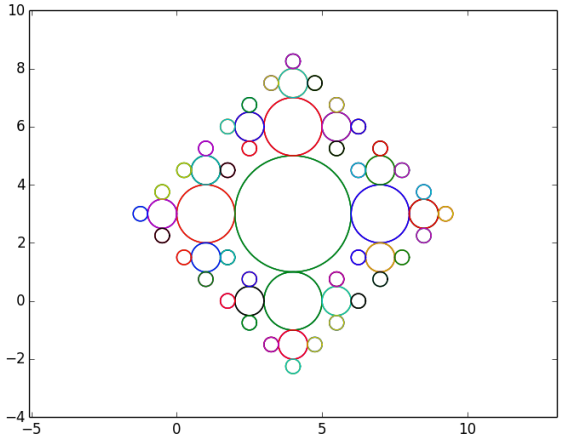
Exercice 11 : cercles

Considérons le dessin suivant, effectué de façon récursive :



La figure est formée d'un cercle et de deux copies de ce cercle ayant subi une réduction d'un facteur 2, ces deux petits cercles étant tangents extérieurement au cercle initial et tels que les lignes des centres sont parallèles aux axes du repère. Ces deux petits cercles deviennent à leur tour "cercle initial" pour poursuivre la figure un nombre donné de fois.

1. Écrire une procédure récursive `CerclesRec(x,y,r,n)` traçant la figure ci-dessus, où le cercle initial est de centre (x, y) et de rayon r, et où on demande n itérations du processus.
   On constatera que les centres des deux cercles définis à partir de celui de centre (x, y) et de rayon r, ont pour centres les points $(x, y - 3r/2)$ et $(x + 3r/2, y)$.
2. Modifier le programme afin d'obtenir la figure suivante :



On pourra utiliser un paramètre supplémentaire `position`*, qui sera une des quatre chaînes de caractères 'haut', 'bas, 'gauche', 'droite'.*