Informatique — MPSI/PCSI, 2023–2024

Correction du TP n° 7

1 Fonction récursive

Exercice 7.1
Version itérative de l'algorithme d'Euclide.

```
def euclide(a,b):
    while b!=0:
        a,b=b,a%b
    return(a)
```

où l'on renvoie forcément a car la condition de terminaison de la boucle `while` est $b = 0$.

Sachant que pour tout $(a, b) \in \mathbb{N}_*^2$, $a > b$, on a :

$$\text{PGCD}(a, b) = \text{PGCD}(a - b, b) = \text{PGCD}(b, a\%b)$$

où $a\%b$ désigne le reste de la division de a par b, il vient alors la version récursive de l'algortithme d'Euclide :

$$\text{PGCD}(a, b) = \begin{cases} a & \text{si } b = 0 \\ \text{PGCD}(b, a\%b) & \text{sinon} \end{cases}$$

que l'on peut implémenter sous la forme :

```
def PGCD(a,b):
    if b==0:
        return(a)
    else:
        return(PGCD(b,a%b))
```

Exercice 7.2
Le seul point auquel il faut être attentif est l'ordre des instructions : on appelle la fonction à l'indice $n - 1$ puis on affiche une nouvelle ligne dans le premier cas, on affiche une nouvelle ligne puis on affiche la fonction dans le second.

1. Pour afficher des étoiles dans l'ordre décroissant, il suffit

$$\forall n \in \mathbb{N}_*, \quad \text{étoiles}(n) = \begin{cases} \text{Afficher } \texttt{"*"} & \text{si } n = 1 \\ \text{Afficher } n \texttt{"*"} \text{ puis appeler étoiles}(n-1) & \text{sinon} \end{cases}$$

avec le variant n qui décroit de n à 1. ce que l'on implémente sous la

```
def etoiles(n):
    print(n*"*")
    if n>1:
        etoiles(n-1)
```

La pile d'éxécution pour $n = 4$ conduit à :

```
etoile(4)
    ****
    etoiles(3)
        ***
        etoiles(2)
            **
            etoiles(1)
                *
```

2. En remarquant qu'en appelant la fonction avant d'afficher, soit avec :

$$\forall n \in \mathbb{N}_*, \quad \text{étoiles}(n) = \begin{cases} \text{Afficher } \texttt{"*"} & \text{si } n = 1 \\ \text{appeler étoiles}(n-1) \text{ puis Afficher } n \texttt{"*"} & \text{sinon} \end{cases}$$

avec le même variant, alors la pile d'exécution devient :

```
etoile(4)
    etoiles(3)
        etoiles(2)
            etoiles(1)
                *
            **
        ***
    ****
```

et produit bien le nombre croissant d'étoiles attendu.

Exercice 7.3
Soient a et b deux entiers non nuls avec $a > b$. Alors :

$$\exists q \in \mathbb{N}, \exists r \in [\![0, b-1]\!], a = qb + r$$

d'où :

$$\begin{aligned} a &= qb + r \\ a \leftarrow a - b &= (q-1)\,b + r \\ &\vdots \\ a \leftarrow a - b &= (q-(q-1))\,b + r \geqslant b \\ a \leftarrow a - qb &= (q-q)\,b + r = r < b \end{aligned}$$

1. On en déduit que le reste de la division euclidienne de a par $b \in [\![1, a-1]\!]$ est défini de façon récursive par :

$$R(a,b) = \begin{cases} a & \text{si } a < b \\ R(a-b,b) & \text{sinon} \end{cases}$$

avec le variant « a » qui décroit de a à r par pas de $-b$.

```
def R(a,b):
    if a<b:
        return(a)
    else:
        return(R(a-b,b))
```

Pour $a = 11$ et $b = 3$, la pile d'exécution est :

```
R(11,3)
    R(8,3)
        R(5,3)
            R(2,3)
                2
            2
        2
    2
```

2. En observant que le nombre d'appels récursifs correspond au quotient q, il suffit pour définir le quotient d'utiliser avec $a = 11$ et $b = 3$, une pile d'exécution de la forme :

```
Q(11,3)
    1+Q(8,3)
```

```
        1+Q(5,3)
            1+Q(2,3)
                0
            1
        2
    3
```

telle que la fonction quotient soit alors définie par :

$$Q(a,b) = \begin{cases} 0 & \text{si } a < b \\ 1 + Q(a-b,b) & \text{sinon} \end{cases}$$

que l'on implémente sous la forme :

```
def Q(a,b):
    if a<b:
        return(0)
    else:
        return(1+Q(a-b,b))
```

Exercice 7.4

```
def permut(chaine):
    """Génère la liste des permutations d'une chaîne de caractères distincts.

    Paramètres :
        chaine : str
        la chaine de caractère dont on génère les permutations.
    Retourne :
        list
        liste des permutations.
    """

    l=len(chaine)
    # Si la chaîne est vide, on retourne la chaîne
    if l==0:
        return [chaine]
    # Sinon, on isole le premier caractère et on génère récursivement
    # toutes les permutations du reste de la chaîne
    else:
        L=[]
        c=chaine[0]
        Li=permut(chaine[1:])
        # Pour chaque permutation obtenue, on génère toutes les permutations de la chaîne
initiale
        # obtenues en ajoutant le premier caractère a un endroit donné de la permutation
        for p in Li:
            for i in range(l):
                L.append(p[:i]+c+p[i:])
        return L
```

ce que l'on peut implémenter directement comme :

```
def permutations(C):
    if len(C)==1:
        return(C)
    else:
        return([T[:i]+C[0]+T[i:] for i in range(len(C)) for T in permutations(C[1:])])
```

2 Généralisation

Exercice 7.5
Pour $n = 13$, on appelle la fonction successivement pour $n = 6$, $n = 3$, $n = 1$ et $n = 0$. Il faut deux multiplications pour calculer `exp_rapide(x,n)`: à partir de `exp_rapide(x,m)`: (où `m=n//2`) si n est impair et une multiplication si n est pair. On effectue donc au total $2 + 1 + 2 + 2 = 7$ multiplications.

En comparaison, l'algorithme de multiplication naïf demanderait à effectuer 12 multiplications.

Exercice 7.6
On commence par tester si la liste est vide : si oui, x n'appartient pas à L. Sinon, on calcule $x = \llcorner \frac{n}{2} \lrcorner$ et on sépare les cas $x = x_m$ (on a trouvé x et on retourne `True`), $x_m > x$ (on appelle récursivement la fonction en passant en argument la liste $[x_0, \ldots, x_{m-1}]$) et $x_m < x$ (appel récursif avec $[x_{m+1}, \ldots, x_{n-1}]$).

```
def rech_dicho(x,L):
    """Cherche si x appartient a L par dichotomie

    Parametres
    x (float)
    L (list) : liste de réeles tries par ordre croissant

    Retour
    Boolean
    True si x appartient a L, False sinon
    """
    n=len(L)
    if n==0:
        return False
    else:
        m=n//2
        if L[m]==x:
            return True
        elif L[m]>=x:
            return rech_dicho(x,L[:m])
        else:
            return rech_dicho(x,L[m+1:])
```

* *
*