Informatique PCSI
Corrigé TP 5

Exercice 1

Deux versions : dans la première, un seul paramètre est modifié, dans la seconde les deux. Dans la première version, les additions (`+ 1`) sont en attente des retours des appels récursifs.

```
def somme_rec(a, b):
    if b == 0:
        return a
    else:
        return somme_rec(a, b-1) + 1

def somme_rec(a, b):
    if b == 0:
        return a
    else:
        return somme_rec(a+1, b-1)
```

Le nombre d'appels récursifs est égal à `b`. On peut donc réfléchir à permuter `a` et `b`.

Exercice 2

Deux versions comme dans l'exercice précédent avec un nombre d'appels récursifs égal à `b`. :

```
def somme_rec1(a, b):
    if b == 0:
        return a
    elif b > 0:
        return somme_rec1(a, b-1) + 1
    else:
        return somme_rec1(a, b+1) - 1


def somme_rec2(a, b):
    if b == 0:
        return a
    elif b > 0:
        return somme_rec2(a+1, b-1)
    else:
        return somme_rec2(a-1, b+1)
```

Exercice 3

```
def pair(n):
    if n > 0:
        return pair(n-2)
    else:
        return n == 0
```

Exercice 4

Une fonction `puissance` récursive :

```
def puissance(x, n):
    if n == 0:
        return 1
    else:
        return x * puissance(x, n-1)
```

On peut visualiser les appels récursifs et les résultats renvoyés :

```
def puissance(x, n, ch=""):
    if n == 0:
        return 1
    else:
        ch = ch + "--"
        param = str(x) + ", " + str(n-1)
        print(ch + "> appel de puissance(" + param + ")")
        y = puissance(x, n-1, ch)
        print(ch + "> résultat de puissance(" + param + "): " + str(y))
        return x * y
```

```
>>> puissance(2, 4)
--> appel de puissance(2, 3)
----> appel de puissance(2, 2)
------> appel de puissance(2, 1)
--------> appel de puissance(2, 0)
--------> résultat de puissance(2, 0): 1
------> résultat de puissance(2, 1): 2
----> résultat de puissance(2, 2): 4
--> résultat de puissance(2, 3): 8
16
```

Exercice 5

1. Une fonction `puissance1` :

```
def puissance1(x, n):
    if n == 0:
        return 1
    else:
        p = puissance1(x, n//2)
        if n % 2 == 0:
            return p * p
        else:
            return x * p * p
```

2. Une fonction `puissance2` :

```
def puissance2(x, n):
    if n == 0:
        return 1
    else:
        if n % 2 == 0:
            return puissance2(x**2, n//2)
        else:
            return x * puissance2(x**2, n//2)
```

Exercice 6

1. Deux exemples d'écriture d'une fonction `somme`.

```
def somme1(liste):
    if liste == []:
        return 0
    else:
        return liste[0] + somme1(liste[1:])

def somme2(liste):
    if liste == []:
        return 0
    else:
        x = liste.pop() # la liste est modifiée et vide à la fin !
        return somme2(liste) + x
```

2. Avec la première fonction, une liste est construite à chaque appel récursif. Le coût est quadratique en la longueur de la liste. Avec la deuxième fonction, le coût de la fonction `pop` est constant et la liste est modifiée en place. Le coût total est donc linéaire en la longueur de la liste mais la liste est vide après l'exécution de la fonction.

3. Exécution de la première fonction avec le paramètre `[4, 7, 2]` :
- > en attente, doit renvoyer : `4 + somme([7, 2])`
- - - > en attente, doit renvoyer : `4 + (7 + somme([2]))`
- - - - - > en attente, doit renvoyer : `4 + (7 + (2 + somme([])))`
- - - - - - - > arrêt des appels récursifs
- - - - - - - > renvoi de `somme([])` : 4 + (7 + (2 + 0))
- - - - - > renvoi de `(2 + 0)` : 4 + (7 + 2)
- - - > renvoi de `(7 + 2)` : 4 + 9
- > renvoi de `13`