Chapitre : Fonctions récursives – Exercices

Exercice 1 – Récursion comme remplacement d'une boucle

On souhaite demander à l'utilisateur un entier strictement positif.

1.a Version itérative (rappel)

```
n = int(input("Donner un entier positif : "))
while n <= 0:
    print("Erreur")
    n = int(input("Donner un entier positif : "))
```

Questions :

1. Développer la version récursive de ce code.
2. Quel est le cas de base ?
3. Pourquoi la fonction termine-t-elle ?
4. En quoi cette fonction remplace-t-elle la boucle while ?

Exercice 2 – Compter récursivement (récursion simple)

```
def compte(n):
    if n == 0:
        return 0
    return 1 + compte(n-1)
```

Questions :

1. Calculer compte(3)
2. Que calcule cette fonction ?
3. Écrire une version itérative
4. Identifier le cas de base et le cas récursif
5. Développer la version itérative de cette fonction

Exercice 3 – Somme des entiers (classique)

```
def somme(n):
    if n == 0:
        return 0
    return n + somme(n-1)
```

Questions :

1. Calculer somme(4)
2. Donner l'expression mathématique calculée
3. Pourquoi la fonction s'arrête-t-elle ?
4. Que se passe-t-il si n est négatif ?

Exercice 4 – Récursion sur une liste (simple et concrète)

```
def somme_liste(L):
    if L == []:
        return 0
    return L[0] + somme_liste(L[1:])
```

Questions :

1. Quel est le cas de base ?
2. Tracer l'exécution pour L = [1, 3, 5]
3. Pourquoi la taille de la liste diminue-t-elle à chaque appel ?
4. Donner une version avec une boucle for

Exercice 5 – Recherche d'un élément dans une liste (récursive)

```
def contient(x, L):
    if L == []:
        return False
    if L[0] == x:
        return True
    return contient(x, L[1:])
```

Questions :

1. Tester la fonction avec x = 3, L = [1, 2, 3, 4]
2. Identifier le cas de base
3. Pourquoi cette fonction termine-t-elle toujours ?
4. Comparer avec une version itérative

Récursion multiple (consolidation)

Exercice 6 – Fibonacci (rappel fondamental)

```
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)
```

Questions :

1. Calculer fib(4)
2. Combien y a-t-il d'appels récursifs ?
3. Pourquoi parle-t-on de récursion multiple ?
4. Quel est le problème principal de cette fonction ?

Exercice 7 – Nombre de façons de monter un escalier

Un escalier a n marches.
On peut monter 1 ou 2 marches à la fois.

```
def nb_facons(n):
  if n == 0:
    return 1
  if n < 0:
    return 0
  return nb_facons(n-1) + nb_facons(n-2)
```

Questions :

1. Calculer nb_facons(3)
2. Pourquoi y a-t-il deux appels récursifs ?
3. Faire le lien avec la suite de Fibonacci
4. Identifier les cas de base

Exercice 8 – Récursion multiple contrôlée

```
def f(n):
  if n == 0:
    return 1
  return f(n-1) + f(n-1)
```

Questions :

1. Calculer f(1), f(2), f(3)
2. Que vaut f(n) en fonction de n ?
3. Combien d'appels récursifs pour f(3) ?
4. Pourquoi cette fonction devient rapidement lente ?

Exercice 9 – Erreur classique (à corriger)

```
def puissance(n):
    if n == 0:
        return 1
    puissance(n-1)
```

Questions :

1. Quel est le problème dans cette fonction ?
2. La fonction termine-t-elle ?
3. Corriger la fonction pour calculer 2^n

Exercice 10 – Comparer récursif / itératif (objectif bac)

```
def factorielle_rec(n):
    if n == 0:
        return 1
    return n * factorielle_rec(n-1)
```

Questions :

1. Écrire la version itérative
2. Identifier le cas de base
3. Dans quel cas la récursion est-elle plus lisible ?
4. Dans quel cas la boucle est-elle préférable ?