Devoir sur table
Récursivité

Durée de l'épreuve : 1h
L'usage de la calculatrice n'est pas autorisé.

Exercice 1 (2 points)

La somme des n premiers entiers

Pour définir la somme des n premiers entiers, on a l'habitude d'écrire la formule :

$$0+1+2+3+4+...+n$$

Vous allez écrire la fonction récursive somme(n) qui calcule la somme des n premiers entiers.

Consigne : Recopiez le code suivant et complétez .... sur votre copie :

```
def somme(n) :
    if n == ....:
        return 0
    else :
        return n + ...
```

Exercice 2 (2 points)

Le calcul de la factorielle d'un nombre est une opération mathématique qui consiste à multiplier ce nombre par tous les entiers positifs inférieurs ou égaux à lui. La factorielle d'un nombre entier positif n est notée n!

La **factorielle** d'un nombre n, notée $n!$, est le produit des entiers de 1 à n. Par exemple :

- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
- $0! = 1$ (par définition)

Nous pouvons donc utiliser la définition récursive suivante :

- Si $n = 0$, alors $n! = 1$ (cas de base).
- Sinon, $n! = n \times (n-1)!$ (cas récursif).

Vous allez écrire la fonction récursive appelée factorielle(n) qui prend un entier positif n en paramètre et retourne n!

Consigne : Recopiez le code suivant et complétez .... sur votre copie :

```
def factorielle(n):
    if n == 0:
        return ....
    else:
        return n * ....
```

Exercice 3 (4 points)

1. Voici une fonction codée en Python :

```
def f(n):
    if n == 0:
        print("Partez!")
    else:
        print(n)
        f(n-1)
```

   a. Qu'affiche la commande `f(5)` ?

   b. Pourquoi dit-on de cette fonction qu'elle est récursive ?

2. On rappelle qu'en python l'opérateur + a le comportement suivant sur les chaînes de caractères :

```
>>> S = 'a'+'bc'
>>> S
'abc'
```

   Et le comportement suivant sur les listes :

```
>>> L = ['a'] + ['b', 'c']
>>> L
['a', 'b', 'c']
```

   On a besoin pour les questions suivantes de pouvoir ajouter une chaîne de caractères `s` en préfixe à chaque chaîne de caractères de la liste `liste`. On appellera cette fonction `ajouter`.

   Par exemple, `ajouter("a",["b","c"])` doit retourner `["ab","ac"]`.

   a. Recopiez le code suivant et complétez `.....` sur votre copie :

```
def ajouter(s, liste):
    res = []
    for m in liste:
        res.....
    return res
```

   b. Que renvoie la commande `ajouter("b", ["a","b","c"])` ?

   c. Que renvoie la commande `ajouter("a", [""])` ?

3. On s'intéresse ici à la fonction suivante écrite en Python où `s` est une chaîne de caractères et `n` un entier naturel.

```
def produit(s, n):
    if n == 0:
        return [""]
    else:
        res = []
        for i in range(len(s)):
            res = res + ajouter(s[i], produit(s, n-1))
        return res
```

   a. Que renvoie la commande `produit("ab", 0)` ? Le résultat est-il une liste vide ?

   b. Que renvoie la commande `produit("ab", 1)` ?

   c. Que renvoie la commande `produit("ab", 2)` ?

Exercice 4 (2 points)

Les deux fonctions f1_rec et f2_rec ci-dessous font les mêmes opérations, mais l'instruction print() est placée différemment.

Qu'affiche f(4) ?

```
def f1_rec(i):
    if (i == 8):
        print('cas de base:', i)
    else:
        print(i)
        f1_rec(i+1)

def f2_rec(i):
    if (i == 8):
        print('cas de base:', i)
    else:
        f2_rec(i+1)
        print(i)

def f(n):
    f1_rec(n)
    print('----')
    f2_rec(n)
```