Récursivité

Calculs de sommes

1. On souhaite écrire une fonction **`somme(n)`** qui calcule et renvoie la somme des entiers compris entre 1 et n. On note S_n cette somme :

$$S_n = \sum_{k=1}^{n} k = 1+2+3+\cdots+(n-1)+n$$

On remarque que $1+2+3+\cdots+(n-1) = S_{n-1}$, on a donc :

$$S_n = S_{n-1} + n$$

Tester le code suivant :

```
def somme(n):
    # On suppose n>=1
    if n==1: # (Cas de base)
        return 1
    else: # (n>1 : appel récursif)
        return n+somme(n-1)

print(somme(1))
print(somme(4))
```

2. Quels sont les appels successifs à la fonctions **`somme`** qui sont effectués quand on lance l'instruction **`print(somme(3))`** ? *Réponse :* _____

3. Tester maintenant **`print(somme(0))`** et observer le résultat.
Réponse : _____

4. On peut convenir que $S_0 = 0$. Modifier la fonction **`somme`** pour qu'elle fonctionne également lorsque $n = 0$.

5. Tester avec les instructions :

```
print(somme(0))
print(somme(1))
print(somme(4))
```

Réponse : _____

6. Sur le même principe, écrire une fonction *récursive* **`somme_carres(n)`** qui calcule la somme des carrés des nombres entiers de 1 à n :

$$1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2$$

On rédigera la fonction de manière à ce que le cas $n = 0$ soit pris en compte, on convient naturellement que la somme est nulle pour $n = 0$.

7. Noter le résultat obtenu avec **`print(somme_carres(5))`**. *Réponse :* _____

La même chose avec une liste

8. On souhaite écrire une fonction **`somme_termes(L,n)`** qui calcule et renvoie la somme des éléments de la liste L de l'indice 0 jusqu'à l'indice n, c'est à dire la somme :

$$L[0] + L[1] + L[2] + \cdots + L[n-1] + L[n]$$

Il faudra pour cela supposer que la liste **`L`** considérée contient au moins $n+1$ éléments. On remarque que cette somme peut s'écrire :

$$\underbrace{L[0] + L[1] + L[2] + \cdots + L[n-1]}_{\text{somme de l'indice 0 jusqu'à } n-1} + L[n]$$

Recopier le code suivant et compléter les `???` :

```
def somme_termes(L,n):
    if n==0:
        return ???
    else:
        return ???
```

puis tester avec les instructions :

```
print(somme_termes([1,2,3],0))
print(somme_termes([1,2,3],1))
print(somme_termes([1,2,3],2))
```

Réponse : _____

9. Sur le même principe, écrire une fonction *récursive* **`somme_carres(L,n)`** qui calcule la somme $L[0]^2 + L[1]^2 + \cdots + L[n]^2$. Tester avec les instructions :

```
print(somme_carres([1,2,3],0))
print(somme_carres([1,2,3],1))
print(somme_carres([1,2,3],2))
```

Réponse : _____

Autres calculs récursifs

10. Sur le même principe que les calculs de sommes dans la première partie. Écrire une fonction *récursive* **`factorielle(n)`** qui calcule $n!$ définie par :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

On rédigera la fonction de manière à ce que le cas $n = 0$ soit pris en compte, on rappelle que $0! = 1$.

11. Noter le résultat obtenu avec **`print(factorielle(5))`**. *Réponse :* _____

12. On considère la suite (p_n) définie par :

$$p_n = \frac{2 \times 4 \times 6 \times \cdots \times (2(n-1)) \times (2n)}{1 \times 3 \times 5 \times \cdots \times (2n-1) \times (2n+1)}$$

Trouver une relation de la forme : $p_n = ??? \times p_{n-1}$. *Réponse :* _____

13. En utilisant cette relation, écrire une fonction *récursive* **`p(n)`** qui renvoie la valeur de p_n. On convient que $p_0 = 1$.

14. Noter le résultat obtenu avec **`print(p(4))`**. *Réponse :* _____

Autres calculs récursifs sur des listes

15. On souhaite déterminer s'il y a un éléments positif ($\geqslant 0$) parmi les n premiers termes d'une liste **`L`** (qui doit donc contenir au moins n termes). On renverra **`True`** s'il y a un élément positif et **`False`** dans le cas contraire. On note les points suivants :

- Si $n = 0$, on doit renvoyer **`False`** ;
- Si $n \geqslant 1$ et si **`L[n-1]`** $\geqslant 0$, on doit renvoyer **`True`** ;
- Si $n \geqslant 1$ et si **`L[n-1]`** < 0, on doit chercher s'il y a un élément positif parmi les $n-1$ premiers éléments de **`L`**.

Écrire une fonction *récursive* **`element_positif(L,n)`** qui met en œuvre ces principes. Tester avec les instructions :

```
print(element_positif([-1,-2,3],0))
print(element_positif([-1,-2,3],2))
print(element_positif([-1,-2,3],3))
```

Réponse : _____

16. Écrire une fonction *récursive* **nombre_de_positifs(L,n)** qui renvoie le nombre d'éléments positifs parmi les n premiers éléments de la liste **L**.

```
print(nombre_de_positifs([-1,2,3],0))
print(nombre_de_positifs([-1,2,3],1))
print(nombre_de_positifs([-1,2,3],2))
```

Réponse : _____

Récursivité sur des listes, autre approche

17. En général, pour écrire une fonction récursive sur une liste **L**, on n'utilise pas un paramètre **n** indiquant le nombre de termes à considérer mais on modifie la liste elle-même au cours des appels récursifs. Par exemple, pour définir une fonction *récursive* **somme(L)** qui calcule la somme de tous les termes de la liste **L**, on peut procéder de la manière suivante :

- Le cas de base est celui où la liste **L** est vide, on renvoie alors 0 ;
- Si la liste **L** n'est pas vide, on retire son dernier élément et on le note **x** à l'aide de l'instruction **x=L.pop()**. On calcule ensuite la somme **s** des termes qui restent par un appel récursif **s=somme(L)**. On peut alors remettre **x** dans **L** à l'aide de l'instruction **append** puis renvoyer le résultat.

Écrire la fonction *récursive* **somme(L)** qui correspond à cette description. Tester avec les instructions :

```
print(somme([]))
print(somme([1]))
print(somme([1,2]))
```

Réponse : _____

18. Pour vérifier que la liste **L** n'a pas été modifiée lors de l'appel, tester les instructions :

```
L = [1,2,3]
s = somme(L)
print(L)
```

Réponse : _____

19. Sur le même principe, écrire une fonction *récursive* **produit(L)** qui calcule le produit des éléments de la liste **L**. Tester avec les instructions **print(produit([]))** et **print(produit([1,2]))**.

Réponse : _____

20. Sur le même principe, écrire une fonction *récursive* **longueur(L)** qui renvoie le nombre d'éléments contenus dans la liste **L**.
Tester avec les instructions **print(longueur([]))** et **print(longueur([1,2]))**.
Bien entendu, on n'utilisera pas la fonction **len** présente dans PYTHON.

Corrections

Q 5.

```
def somme(n):
    if n==0: # (Cas de base)
        return 0
    else: # (n>0 : appel récursif)
        return n+somme(n-1)

print(somme(0))
print(somme(1))
print(somme(4))
```

```
0 1 10
```

Q 6.

```
def somme_carres(n):
    if n==0: # (Cas de base)
        return 0
    else: # (n>0 : appel récursif)
        return n**2+somme(n-1)
```

Q 7.

```
print(somme_carres(5))
```

```
35
```

Q 8.

```
def somme_termes(L,n):
    if n==0:
        return L[0]
    else:
        return L[n]+somme_termes(L,n-1)

print(somme_termes([1,2,3],0))
print(somme_termes([1,2,3],1))
print(somme_termes([1,2,3],2))
```

```
1 3 6
```

Q 9.

```python
def somme_carres(L,n):
    if n==0:
        return L[0]**2
    else:
        return L[n]**2+somme_carres(L,n-1)

print(somme_carres([1,2,3],0))
print(somme_carres([1,2,3],1))
print(somme_carres([1,2,3],2))
```

```
1 5 14
```

Q 10.

```python
def factorielle(n):
    if n==0: # (Cas de base)
        return 1
    else: # (n>0 : appel récursif)
        return n*factorielle(n-1)
```

Q 11.

```python
print(factorielle(5))
```

```
120
```

Q 12.

$$p_n = \frac{2n}{2n+1} p_{n-1}$$

On note que par définition :

$$p_1 = \frac{2}{3} = \frac{2}{3} p_0$$

et ceci est cohérent avec le fait de prendre $p_0 = 1$.

Q 13.

```python
def p(n):
    if n==0:
        return 1
    else:
        return 2*n/(2*n+1)*p(n-1)
```

Q 14.

```python
print(p(4))
```

```
0.4063492063492063
```

Q 15.

```
def element_positif(L,n):
    if n==0:
        return False
    else:
        if L[n-1]>=0:
            return True
        else:
            return element_positif(L,n-1)

print(element_positif([-1,-2,3],0))
print(element_positif([-1,-2,3],2))
print(element_positif([-1,-2,3],3))
```

```
False False True
```

Q 16.

```
def nombre_de_positifs(L,n):
    if n==0:
        return 0
    else:
        c = nombre_de_positifs(L,n-1)
        if L[n-1]>=0:
            c = c+1
        return c

print(nombre_de_positifs([-1,2,3],0))
print(nombre_de_positifs([-1,2,3],1))
print(nombre_de_positifs([-1,2,3],2))
```

```
0 0 1
```

Q 17.

```
def somme(L):
    if L==[]:
        return 0
    else:
        x = L.pop()
        s = somme(L)
        L.append(x)
        return s+x

print(somme([]))
print(somme([1]))
print(somme([1,2]))
```

```
0 1 3
```

Q 19.

```
def produit(L):
    if L==[]:
        return 1
    else:
        x = L.pop()
        p = produit(L)
        L.append(x)
        return p*x

print(produit([]))
print(produit([1,2]))
```

```
1 2
```

Q 20.

```
def longueur(L):
    if L==[]:
        return 1
    else:
        x = L.pop()
        l = longueur(L)
        L.append(x)
        return l+1

print(longueur([]))
print(longueur([1,2]))
```

```
1 3
```