

TP4 – Récursivité

1 | Sommer les termes d'une liste

Soit `L` une liste de nimbre. On souhaite que `somme_termes(L)` renvoie la somme des nombres de `L` sans modifier la liste. La somme d'une liste vide vaut 0.

Question 1 : Compléter la chaîne d'égalités ci-dessous, puis faire la remontée en déterminant le résultat de chaque appel.

```
somme_termes([3, 2, 5]) = 3 + somme_termes([2, 5])
somme_termes([2, 5])    = ...
somme_termes([5])       = ...
somme_termes([])        = ...
```

Question 2 : Écrire la fonction récursive `somme_termes(L)`, sans boucle ni `sum`. Préciser le cas de base. Vérifier le résultat de la question précédente.

2 | Le plus petit élément

On rappelle que l'instruction `L.pop(i)` permet de supprimer l'élément d'indice i de la liste L.
Soit `L` une liste non vide. La fonction `plusPetit` compare les deux premiers éléments, supprime le plus grand des deux (ou l'un des deux s'ils sont égaux) et recommence sur la liste raccourcie.

```
def plusPetit(L):
    if CONDITION_1:
        return L[0]
    if CONDITION_2:
        L.pop(1)
    else:
        L.pop(0)
    return plusPetit(L)
```

Question 3 : Compléter les deux conditions. Justifier que l'élément supprimé n'est pas indispensable pour conserver le minimum.

Question 4 : Prévoir la valeur affichée dans la console par les instructions suivantes. Examiner le contenu de `L1` après l'appel.

```
L1 = [3, 2, 5, 9, 7, 12, 1, 5, 8, 6, 10]
print(plusPetit(L1))
```

Question 5 : Exécuter de nouveau `L1 = [3, 2, 5, 9, 7, 12, 1, 5, 8, 6, 10]`, puis tester `plusPetit(L1[:])`. Expliquer pourquoi cette fois `L1` reste intacte.

3 | ASCII art

Question 6 : Écrire une fonction récursive `triangle` qui prend en argument `caractere` une chaîne de caractere non nulle et `hauteur` un entier. Celle-ci affiche dans la console un triangle de hauteur `hauteur` formé de `caractere`.

*Exemple d'appel **dans la console** :*

```
In[0]: triangle('$',3)
$$$
$$
$

In[1]: triangle('*',2)
**
*
```

Question 7 : Écrire la fonction récursive `sablier(caractere, n)` qui affiche, pour $n \geq 1$, des lignes de $n, n-1, \ldots, 1, 2, \ldots, n$ `caractere`. Déterminer quelle ligne s'affiche à la descente des appels et laquelle s'affiche lors des retours.

```
In[3]: sablier('*',3)
***
**
*
**
***
```

4 | Palindrome

Un palindrome se lit de la même façon dans les deux sens. Dans le TP2, exercice 5, on comparait la chaîne saisie à son inverse :

```
def palindrome(chaineCaractere):
    inverse = maStr[::-1]
    if inverse == maStr:
        return True
    else:
        return False
```

Question 8 : Écrire une fonction récursive `palindromeRecur(chaine)` qui renvoie un booléen.

5 | Fractale de cercles tangents

On trace un cercle de centre (x, y) et de rayon r. Puis on trace deux copies réduites : une à droite et une en dessous. Les deux nouveaux cercles ont un rayon $\frac{r}{2}$ et sont tangents extérieurement au cercle de départ. On applique la même règle à chaque nouveau cercle.

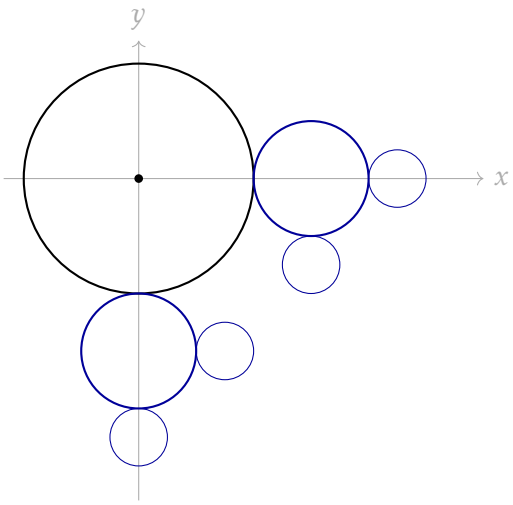


Figure 1 – Profondeur 2 : le cercle de départ, deux enfants et quatre petits-enfants

Question 9 : Déterminer la distance entre le centre du cercle de rayon r et celui d'un enfant pour assurer la tangence. En déduire les deux centres des enfants en fonction de x, y et r.

Copier le code fourni ci-dessous dans un script, *attention à l'indentation*. La fonction `cercle` réalise tout le tracé d'un cercle ; on l'utilise telle quelle.

```
# Importer les outils de tracé de Matplotlib.
import matplotlib.pyplot as plt

# Créer une figure (fig) et une zone de tracé (ax).
fig, ax = plt.subplots()

def cercle(x, y, r):
    # Ajouter dans ax un cercle non rempli, de centre (x, y)
    # et de rayon r. fill=False laisse seulement le contour.
    ax.add_patch(plt.Circle((x, y), r, fill=False, color='b'))

# Tester d'abord avec un seul cercle de rayon 8
cercle(0, 0, 8) # Remplacer ensuite cet appel pour tester cercleRec
# Ajuster les limites des axes aux cercles tracés
ax.autoscale_view()
# Utiliser la même échelle sur x et y
ax.set_aspect("equal")
plt.show()
```

Question 10 : Écrire le fonction récursive `cercleRec(x, y, r, profondeur)`, celle-ci appelle la fonction `cercle` et trace des cercles tel que montré sur la figure 1.

Question 11 : Modifier `cercleRec` pour qu'elle trace quatre enfants à chaque étape, un par direction comme montré par la figure 2.

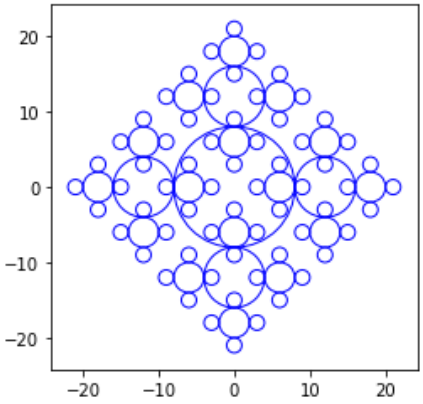


Figure 2 – Des cercles apparaissent dans tous les directions

On souhaite qu'aucun cercle ne soit tracé dans d'autres cercles, comme sur la figure 3. Pour cela, il faut interdire une direction à chaque enfant : si on place un enfant à droite, son parent est à sa gauche et il faut interdire un petit-enfant à gauche.

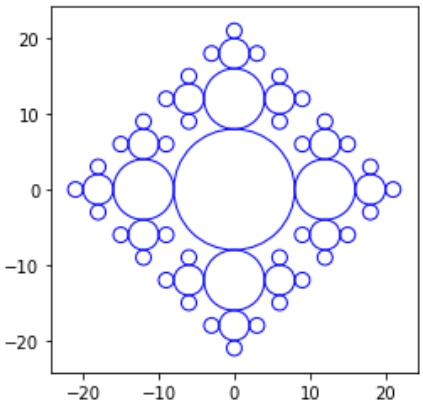


Figure 3 – Avec une direction interdite

Question 12 : Modifier la fonction `cercleRec` pour qu'elle ait ce comportement. *Pour cela, ajouter un paramètre avec la direction interdite.*