TP 06

Fonctions récursives

Savoirs et compétences :

- ❒ *AA.C9 : Choisir un type de données en fonction d'un problème à résoudre*
- ❒ *AA.S12 : Fichiers*

Activité 1 – Quelques éléments d'introduction aux fonctions récurcives

Définitions

Définition Une fonction récursive est une fonction qui s'appelle elle-même.
On appelle récursion l'appel de la fonction à elle-même.

La programmation est un paradigme de programmation au même titre que la programmation itérative. Un programme écrit de manière récursive peut être traduit de manière itérative, même si dans certain cas, cela peut s'avérer délicat.

Méthode
- Une fonction récursive doit posséder une condition d'arrêt (ou cas de base).
- Une fonction récursive doit s'appeler elle-même (récursion).
- L'argument de l'étape de récursion doit évoluer de manière à se ramener à la condition d'arrêt.

▪ **Exemple** Prenons pour exemple du calcul de $n! = \prod_{i=1}^{n} i$ et $n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{si } n \geq 1 \end{cases}$.

```
def fact_it(n : int) -> int :
    res = 1
    for i in range (1,n+1): # ou range(n
        )
        res = res*i
    return res
```

```
def fact_rec(n : int) -> int :
    if n == 0 :
        return 1
    else :
        return n * fact_rec(n-1) # Ré
            cursion
```

▪

Exemples

Suites récurrentes

Soit la suite u_n définie par récurrence pour tout $n \in \mathbb{N}^*$ par $\begin{cases} u_1 = 1 \\ u_{n+1} = \dfrac{u_n + 6}{u_n + 2} \end{cases}$.

```
def un_it (n : int) -> float :
    if n == 1 :
        return 1
    else :
        u = 1
        for i in range(2,n+1):
            u = (u+6)/(u+2)
        return u
```

```
def un_rec (n : int) -> float :
    if n == 1 :
        return 1
    else :
        return (un_rec(n-1)+6)/(un_rec(n
            -1)+2)
```

Recheche dichotomique

```python
def appartient_dicho(e : int , t : list) -> bool:
    """Renvoie un booléen indiquant si e est dans t
    Préconditions : t est un tableau de nombres trié par ordre croissant e est un nombre"""
    g = 0 # Limite gauche de la tranche où l'on recherche e
    d = len(t)-1 # Limite droite de la tranche où l'on recherche e
    while g <= d: # La tranche où l'on cherche e n'est pas vide
        m = (g+d)//2 # Milieu de la tranche où l'on recherche e
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            d = m-1 # On recherche e dans la partie gauche de la tranche
        else:
            g = m+1 # On recherche e dans la partie droite de la tranche
    return False
```

```python
def appartient_dicho_rec(e : int , t : list) -> bool:
    """Renvoie un booléen indiquant si e est dans t
    Préconditions : t est un tableau de nombres trié par ordre croissant e est un nombre"""
    g = 0 # Limite gauche de la tranche où l'on recherche e
    d = len(t)-1 # Limite droite de la tranche où l'on recherche e
    while g <= d: # La tranche où l'on cherche e n'est pas vide
        m = (g+d)//2 # Milieu de la tranche où l'on recherche e
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            d = m-1 # On recherche e dans la partie gauche de la tranche
            appartient_dicho_rec(e,t[g:d])
        else:
            g = m+1 # On recherche e dans la partie droite de la tranche
            appartient_dicho_rec(e,t[g:d])
    return False
```

Activité 2 – Organisation d'une phase de match par poule

On dispose d'une liste de joueurs d'un tournoi de tennis sous phase de poules (exemple : *Master cup*), et on veut créer une liste de matches, de telle sorte que chaque joueur joue contre tous les autres joueurs une seule fois (pas de phase retour).

On peut découper ce problème en sous-problèmes plus simples imbriqués les uns dans les autres :

- On peut ainsi remarquer que si on a une liste de 4 joueurs, on peut résoudre le problème en connaissant une liste de matches pour les 3 premiers joueurs seulement : on prend cette liste, et on y ajoute un match entre le quatrième joueur et chacun des trois autres.
- On peut programmer ce problème avec une fonction récursive dans Python :
  - Soit la liste des joueurs notée "*joueurs*"
  - Il faut commencer par initialiser le problème :
    s'il n'y a qu'un seul joueur, on n'organise aucun match.
  - Il faut donner une relation de récurrence (récurcivité) : on enlève le dernier joueur de la liste, et on demande les matches sans ce dernier.

- Ici il faut penser à ajouter les sous-problèmes en ajoutant un match entre lui et tous les autres joueurs et donc à construire une liste de matches.
- On remet le dernier joueur dans la liste des joueurs, et on renvoie la liste des matches.

On pourra utiliser la fonction .pop(), qui placée après une liste retire son dernier terme :

```
joueurs=list(range(1,5))
joueurs.pop()
4
joueurs
[1, 2, 3]
```

De la même manière, on pourra utiliser la fonction **.append()**, qui placée après une liste ajoute un terme :

```
joueurs=list(range(1,5))
joueurs.append(5)
joueurs
[1, 2, 3, 4, 5]
```

Question 1
Écrire une fonction `matches(joueurs:list)->list` prenant en argument une liste de numéros de joueurs et renvoyant une listes de matchs sous la forme de couple de joueurs.

Activité 3 – Recherche de palindrome

On appelle palindrome un mot qui peut se lire indifféremment de gauche à droite ou de droite à gauche. Ainsi par exemple *radar*, *rotor* ou kayak sont trois palindromes.

Question 2
Écrire une fonction récursive `palindrome(x:str) -> bool` prenant en paramètre une chaîne de caractère `x` et qui retourne `True` si il s'agit d'un palindrome et `False` sinon. (Les espaces seront gérés comme de simples caractères). Pour cela on remarquera que :

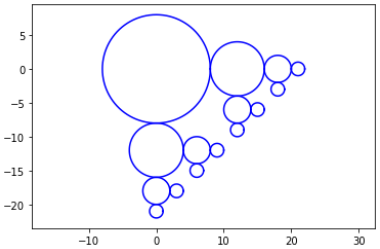
- tout mot de longueur ≤ 1 est un palindrome ;
- un mot est un palindrome si et seulement si ses premier et dernier caractères sont identiques et son sous-mot allant du 2ème caractère jusqu'à l'avant dernier est un palindrome.

Activité 4 – Bubble bobble

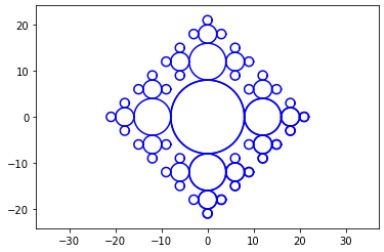
On suppose disposer d'une fonction `circle(coords:list, r:float) -> None` qui trace à l'écran un cercle de centre `coords=(x,y)` et de rayon `r`.

```
import matplotlib.pyplot as plt
import numpy as np
def circle(coords:list, r:float) -> None :
    X, Y = [], []
    for t in range(101):
        X.append(coords[0]+r*np.cos(t*np.pi/50))
        Y.append(coords[1]+r*np.sin(t*np.pi/50))
    plt.plot(X, Y, 'b')
```

Question 3 *Définir la fonction* `bubble1(n:int, x:float, y:float, r:float) -> None` *permettant de tracer la figure suivante.*

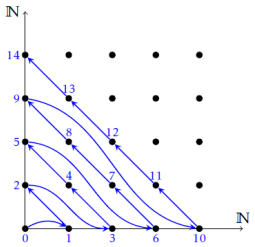


Question 4 *Définir la fonction* `bubble2(n:int, x:float, y:float, r:float, d:str) -> None` *permettant de tracer la figure suivante.*



Activité 5 – Numérotation des points d'un quadrillage

On démontre que l'ensemble $\mathbb{N}\times\mathbb{N}$ est dénombrable en numérotant chaque couple $(x,y)\in\mathbb{N}^2$ suivant le procédé suggéré par la figure ci-dessous.



Question 5 *Rédiger une fonction récursive* `numerote(n:int) -> int` *qui retourne le numéro du point de coordonnées* (x,y).

Question 6 *Rédiger la fonction* `reciproque(n:int) -> tuple`, *là encore de façon récursive*

.