**Durée : 1h.** Les calculatrices sont **interdites**

## Exercice 1 *Carrés*

Un étudiant souhaite programmer une fonction `listecarres(N)` en `Python` qui construit et renvoie une liste $L$ contenant les carrés des entiers de $0$ à $N$.

Il propose le code suivant :

```
def listecarres(N):
    L=[]
    for i in range(N):
        L.append(i**2)
        return(L)
```

Son algorithme est-il correct, et si non, quelle(s) est(sont) les erreurs commises ?

## Exercice 2 *Algorithmes de contrôle*

Une liste $X$ est trié par ordre croissant si $X[i] \leq X[i+1]$ pour tout $i \in [\![0, n-1]\!]$, où $n$ est le nombre d'éléments de $X$.

On va étudier plusieurs algorithmes permettant de tester si une liste est triée.

1. (a) On considère l'algorithme suivant :

```
def testi(X):
    n=len(X)
    res=True
    for i in range(n-1):
        if X[i]>X[i+1]:
            res=False
    return(res)
```

Justifier que cet algorithme est correct, c'est à dire qu'il renvoie `True` si la liste $X$ est triée par ordre croissant, `False` sinon.

   (b) Quel est sa complexité $A_N$ en nombre de comparaisons pour une liste de longueur $N$ ?

2. (a) Si $X$ est une liste de longueur $N \geq 2$, que représentent les commandes Python `X[0:N//2]` et `X[N//2:]` ?

   (b) Elaborer un algorithme récursif `testr(X)` permettant de vérifier qu'un tableau X est trié ou non, pour lequel, si la liste $X$ a une longueur $N$ supérieure ou égale à $2$, on appelle récursivement `testr(X[0:N//2])` et `testr(X[N//2:])`.

   (c) Estimer sa complexité $B_N$, en nombre de comparaisons, pour une liste de longueur $N = 2^p$, avec $p \in \mathbb{N}^*$.

3. Commentaire ?

## Exercice 3 *Méthode numérique vectorielle*

On souhaite résoudre, de manière numérique approchée, un système différentiel de la forme :

$$(E) \quad Y'(t) = f(t, Y(t)),$$

d'inconnue vectorielle $Y : \mathbb{R} \longrightarrow \mathcal{M}_{p,1}(\mathbb{R})$, pour $p \geq 1$ entier fixé.

On rappelle que la commande `np.array` du package `import numpy as np` permet de créer un tableau numpy, en colonnes, de type `array` à partir d'une suite de valeurs contenues dans une liste, ce qui permet d'appliquer les commandes vectorielles du package numpy à ce tableau.

On propose pour cela une fonction `Solut_approch` d'arguments une fonction `f` de $\mathbb{R} \times \mathbb{R}^p$ vers $\mathbb{R}^p$ (liste à $p$ éléments), un vecteur colonne `Y0` de type `array` à $p$ coordonnées, un instant `t0` réel, un paramètre `dt` réel, et un instant `tmax` réel.

```
def Solut_approch(f,Y0,t0=0,dt=1e-3,tmax=10):
   Y,t = np.array(Y0,dtype=float),t0
   Y_arr,t_arr=[Y],[t]
   while t < tmax: # Tant qu'on n'a pas atteint le t maximal d'intégration
      Y = Y + np.array(f(t,Y))*dt   # On calcule la nouvelle valeur de Y
      t = t + dt   # On incrémente le temps
      Y_arr.append(Y)   # Valeur de Y
      t_arr.append(t)   # Valeur de t
   return(np.array(t_arr),np.array(y_arr))   # Renvoi des tableaux concernant t et Y
```

1. Si l'on appelle `t0` l'instant initial, que représente `Y0` ?
2. Si l'on appelle `tmax` l'instant final, que représente `dt` ?
3. Par défaut, `dt=1e-3` et `tmax=10`.
   Combien y aura-t-il d'itérations dans l'algorihme ci-dessus ?
4. Comment s'appelle cette méthode de calcul, pas à pas d'une solution numérique approchée ?
5. Compléter la commande `return` incomplète pour définir la fonction `f_harmonique` associée à l'oscillateur harmonique :

$$x''(t) = -\sin(t)x(t)$$

```
def f_harmonique(t,Y):   # Transformation de l'oscillateur harmonique x''+sin(t) x=0
   x,v = Y   # en système d'équations différentielles
   return [...., .....]   # x' = v  et v' = -sin(t) x (soit v'+x=0 donc x''+x=0)
```

6. On appelerait la fonction de la manière suivante :

```
tps,sol = Solut_approch(f_harmonique,[1,0])
pos= sol[:,0]
vit = sol[:,1]
```

   Que contient `pos` ? Que contient `vit` ?