TP 1| Bases de la programmation sous Python

1. Eléments de base de programmation

1) Ecrire une fonction $f(n)$ qui vous renvoie la liste des nombres pairs de 0 à n si n est pair ou la liste des nombres impairs de 1 à n si n est impair.

```
>>> f(7) renvoie [1,3,5,7]
```

2) Ecrire une fonction $f(x, L)$ où x est un entier et L une liste, et qui vous renvoie $True$ si $x \in L$ et $False$ sinon. On ne cherchera pas à optimiser l'efficacité de cette fonction.

3) Ecrire une fonction $f(L, M)$ où L et M sont 2 listes de même taille et qui vous renvoie la liste de la somme terme à terme des éléments des deux listes :

```
>>> f([1,2,3],[4,5,6]) renvoie [5,7,9]
```

4) Ecrire une fonction $f(\,)$ qui vous demande d'entrer une lettre indéfiniment jusqu'à ce que la lettre entrée soit "q". On pourra utiliser la commande *input()* qui permet à l'utilisateur de rentrer des caractères à l'aide de son clavier. Lorsque "q" est entré, la fonction affiche "Quitter" et termine.

5) Tester l'instruction `"&"*6.` Ecrire une fonction $f(p\,, c)$ affichant un carré creux de côté p avec le caractère c :

```
>>> f(4,"*")
****
*  *
*  *
****
```

6) Créer une fonction $f(L, M)$ prenant en argument 2 listes $[u_x, u_y, u_z]$ et $[v_x, v_y, v_z]$, coordonnées de $\vec{u}$ et $\vec{v}$ dans $(Oxyz)$, et qui vous renvoie le produit scalaire $\vec{u} \cdot \vec{v}$. On rappelle que le produit scalaire de deux vecteurs est défini par $\vec{u} \cdot \vec{v} = u_x v_x + u_y v_y + u_z v_z$

2. Particule soumise à une force constante

On considère une particule de masse m, initialement au repos, soumise à une force constante F. On souhaite établir l'expression de sa vitesse v en fonction du temps t. Constatant que la seconde loi de Newton divergeait pour $t \to \infty$, Einstein a proposé de modifier cette loi.
L'objectif de cette partie est de tracer l'allure des courbes de vitesse $v(t)$ selon le modèle de Newton et le modèle d'Einstein et de comparer leur comportement asymptotique.

Compléments :

- Importer les librairies numpy (renommé **np**) et matplotlib.pyplot (renommé **pl**)
- Quelques structures utiles :

[**2*x + 1 for x in range(10)]**	Renvoie la liste créée par compréhension des valeurs 2*x+1 pour x variant de 0 à 9
np.linspace(a,b,n)	Renvoie n valeurs entre a et b inclus avec un pas égal. La sortie est un tableau de type array
np.arange(a,b,p)	Renvoie le tableau des elements de a à b séparés d'un pas p
pl.plot(x,y)	x et y étant des listes de valeurs, représente l'ensemble des points de coordonnées (x,y)
pl.grid()	Affiche un quadrillage sur la figure obtenue
pl.axis([xmin,xmax,ymin,ymax])	Limite la zone d'affichage
pl.axis("tight")	Ajuste automatiquement la zone d'affichage
pl.xlabel("text") **pl.ylabel("text")**	Affiche un titre pour les axes x et y
pl.show()	Affiche le graphique
pl.legend(["legendes"])	Affiche une légende pour chaque courbe tracée, dans l'ordre des éléments de la liste

1) Montrer qu'en mécanique classique, l'expression de la vitesse en fonction du temps dans la situation étudiée ici s'exprime trivialement par $v = \frac{F}{m}t$.

2) Réaliser une fonction $vN(t)$ permettant d'obtenir la valeur de v d'après le modèle de Newton, pour un paramètre t donné. On prendra $F = 1\ N$ etm $= 1\ kg$.

3) Créer une fonction $courbeN()$ affichant la courbe $t \mapsto vN(t)$ pour $t \in [0{,}10]$ (fixer votre pas ou le nombre de valeurs). Limiter la fenêtre aux valeurs de vN comprises entre 0 et 1.5.

Albert Einstein a proposé de modifier la loi de la façon suivante (notations physiciennes) :

$$\frac{d}{dt}\left(\frac{mv(t)}{\sqrt{1-\frac{v^2(t)}{c^2}}}\right) = F$$

4) Montrer que cette équation conduit à :

$$v(t) = \frac{F.c.t}{\sqrt{c^2m^2 + F^2t^2}}$$

5) Réaliser une fonction $vE(t,c)$ permettant d'obtenir la valeur de v d'après le modèle d'Einstein, pour un paramètre t donné. On prendra : $F = 1\ N, m = 1\ kg$ et $c = 1\ m.s^{-1}$ par défaut.

6) Réaliser une fonction $courbeE(c)$ permettant d'obtenir la représentation graphique de $vE(t,c)$ pour $t \in [0{,}10 \times c]$. On limitera la fenêtre aux valeurs de vE comprises entre 0 et $1.5{\times}c$.

7) Réaliser une fonction $courbe(c)$ permettant d'afficher simultanément les courbes $vN(t)$ et $vE(t,c)$.

8) Faire varier le paramètre c et constater graphiquement la vitesse maximale v_{max} que peut atteindre une particule soumise à une force constante, selon le modèle d'Einstein. Le vérifier en calculant une limite lorsque $t \to +\infty$.

3. Eléments de balistiques et parabole de sécurité

Soit une citadelle située à 10m de hauteur qui envoie un boulet de canon $M(x,z)$ avec une vitesse $v_0 = 20\ m.s^{-1}$ et un angle de tir α quelconque. Le projectile n'est soumis qu'à son poids et toutes forces de frottement ou de poussée d'Archimède seront ici négligées. On pose $h = \frac{{v_0}^2}{2g}$.
On souhaite représenter les trajectoires du boulet pour différentes valeurs de l'angle de tir, déterminer l'angle de portée maximale et représenter la courbe limite de l'ensemble des points accessibles par le boulet (parabole de sécurité).
On souhaite finalement obtenir la valeur de l'angle de portée maximale, les courbes des paraboles de tir pour différents angles de tir et la courbe de la parabole de sécurité.

Compléments :

math.radians(x)	convertit un angle x en radians lorsque celui-ci est transmis en degrés
math.degrees(x)	convertit un angle x en degrés lorsque celui-ci est transmis en radians
np.roots([coeff])	renvoie les zéros d'un polynôme défini par ses coefficients, séparés par des virgules. np.roots([2,1,-3]) résout $2x^2 + x - 3 = 0$

1) Programmer 2 fonctions $x(t, alpha)$ et $z(t, alpha)$ renvoyant les valeurs de x et z pour t et α donnés (ou une seule fonction renvoyant en même temps x et z) à partir de leurs équations horaires : $x(t) = v_0 cos\alpha . t$

$$z(t) = -\frac{1}{2}gt^2 + v_0 sin(\alpha).t + z_0$$

2) Définir une fonction traçant $z = f(x)$ pour α quelconque, à partir de leurs définitions horaires $x(t)$ et $z(t)$.

3) Définir une fonction $impact(alpha)$ permettant d'obtenir numériquement l'instant t_impact de l'impact au sol selon l'angle alpha choisi en paramètre.

4) Utiliser cette nouvelle fonction pour représenter $z = f(x)$ entre l'instant initial et le moment de l'impact uniquement, quel que soit l'angle α entré en paramètre.

5) A partir des fonctions établies précédemment, créer une fonction déterminant l'angle de portée maximale et la valeur de la portée maximale des tirs.

6) Créer une fonction permettant d'obtenir la représentation graphique de toutes les paraboles de tir pour des angles alpha compris entre 0 et 90°.

7) Montrer que l'équation du mouvement du boulet peut se mettre sous la forme :

$$tan^2\alpha - \frac{4h}{x}tan\alpha + (1 + \frac{4h(z - z0)}{x^2}) = 0$$

8) En déduire que l'équation de l'enveloppe de l'ensemble des points accessibles par le projectile est donnée par l'annulation du discriminant de cette équation et a pour équation : $z = h - \frac{x^2}{4h} + z_0$

9) Représenter cette courbe, appelée parabole de sécurité.

10) (Plus dur). A partir des fonctions précédentes, créer une fonction permettant d'obtenir numériquement la valeur minimale de la vitesse, v_{min}, à laquelle doit être tiré un boulet de canon pour pouvoir atteindre un point $M(x_0, z_0)$ donné en paramètre. La fonction renverra v_{min} ainsi que l'angle α_0 du tir. Evaluez la complexité de l'algorithme réalisé.