Vecteur

L'objectif de ce TD est de réaliser une classe permettant de représenter un vecteur 3D. Chaque vecteur possédera donc des composantes x,y et z. On va chercher à implémenter les opération usuelles.

1. Créer une classe Vecteur3D que l'on pourra instancier avec trois attributs (x, y, z)
2. Ecrire la méthode qui renvoie une chaîne de caractère représentant le vecteur. On appellera cette méthode __repr__ (pourquoi ?). Cette méthode ne doit pas imprimer la chaîne de caractère mais la renvoyer.
3. Ecrire une méthode `norm` qui renvoie la norme du vecteur.
4. Modifier la méthode __init__ pour qu'elle puisse fonctionner aussi si un argument est un Vecteur3D, un tuple ou une liste (de longueur 3).
5. Ecrire une méthode `addition` qui prend comme argument un vecteur et renvoie la somme. On vérifiera que l'argument est bien de type `Vecteur3D` et sinon on soulevera l'erreur `NotImplementedError`
6. Il existe en Python des méthodes spéciales dont le nom commence et termine par "__", ces méthodes vont être appelées automatiquement dans certaines circonstances. Par exemple, la méthode `__repr__` est appelée par la fonction `print`. Il existe un série de commande appelée par les opérateurs unitaires ou binaires. Voir la page : https://docs.python.org/3/reference/datamodel.html#emulating-numeric-types

   On va écrire des méthodes pour que

   (a) '+' soit la somme vectoriel (`__add__`)

   (b) '-' soit la différence (`__sub__`)

   (c) '-' (opérateur unitaire, `__neg__`) soit l'opposé

   (d) '*' soit le produit scalaire si on a deux vecteurs et la multiplication par un scalaire si on a un nombre (`__mul__`)

   (e) '/' division par un nombre (`__div__`)

   (f) '**' soit le produit vectoriel (`__pow__`)

   (g) '==' teste si deux vecteurs sont identiques (`__eq__`)

   A chaque fois, il faut tester si le type de donnée est le bon et sinon renvoyer la constante `NotImplemented`.

   Il est important de toujours tester son code. En supposant que le test d'égalité fonctionne, écrire des tests pour chacune des méthodes ci-dessus. On utilisera l'instruction `assert`. Ce test sera dans un fichier indépendant qui importera la librairie dans laquelle est définie `Vecteur3D`. Par exemple

```
u = Vecteur3D(1, 2.3, 5)
v = Vecteur3D(0, -7.3, 3.5)
assert u+v==Vecteur3D(0, -5, 8.5)
```

7. Dans les méthodes précédentes, la multiplication par un scalaire va fonctionner dans le cas `v*a` ou `v` est le vecteur et `a` le scalaire. Que se passe-t-il si on fait `a*v` ? Pourquoi ? La solution passe par la méthode `__rmul__` que vous implémenterez.

8. Ecrire une méthode qui calcule les coordonnées sphérique du vecteur. Ecrire des propriétés telle que si `v` est un vecteur, `v.r`, `v.theta` et `v.phi` renvoie les coordonnées sphérique du vecteur. On pourra utiliser la fonction `atan2` du module `math`.

Point

On représentera un point de façon similaire à un vecteur, sauf que les opérations permises seront différentes. On peut principalement faire la différence de deux points (ce qui donne un vecteur) et la somme d'un point et d'un vecteur (ce qui donne un point)

Ligne

On représente une ligne par une point et un vecteur. Ecrire une méthode qui permet de tester si un point est sur une ligne de telle sorte que `point in ligne` fonctionne (le nom de la méthode est `__contains__`). Quelle erreur faut-il renvoyer si `point` n'est pas un `Point3D` ?

Autre chose

1. Ecrire une fonction qui résout une équation du second degré. Cette fonction renverra toujours deux solutions réelles (éventuellement identiques) ou une erreur.
2. Ecrire une fonction qui renvoie les deux points d'intersection entre une ligne et une sphère (ou une erreur si la ligne passe à côté !).