Python Objet – TP 2

Exercice 1 – Cercle et cône

Définissez une classe **Cercle()**. Les objets construits à partir de cette classe seront des cercles de tailles variées. En plus de la méthode constructeur (qui utilisera donc un paramètre **rayon**), vous définirez une méthode **surface()**, qui devra renvoyer la surface du cercle.

Définissez ensuite une classe **Cylindre()** dérivée de la précédente. Le constructeur de cette nouvelle classe comportera les deux paramètres **rayon** et **hauteur**. Vous y ajouterez une méthode **volume()** qui devra renvoyer le volume du cylindre (rappel : volume d'un cylindre = surface de section × hauteur).

Exemple d'utilisation de cette classe :

```
>>> cyl = Cylindre(5, 7)
>>> print(cyl.surface()) 78.54
>>> print(cyl.volume()) 549.78
```

Ajoutez une classe **Cone()**, qui devra dériver cette fois de la classe **Cylindre()**, et dont le constructeur comportera lui aussi les deux paramètres **rayon** et **hauteur**. Cette nouvelle classe possédera sa propre méthode **volume()**, laquelle devra renvoyer le volume du cône (rappel : volume d'un cône = volume du cylindre correspondant divisé par 3).

Exemple d'utilisation de cette classe :

```
>>> co = Cone(5,7)
>>> print(co.volume()) 183.26
```

Exercice 2

Définissez une classe **JeuDeCartes()** permettant d'instancier des objets dont le comportement soit similaire à celui d'un vrai jeu de cartes. La classe devra comporter au moins les quatre méthodes suivantes :

- méthode constructeur : création et remplissage d'une liste de 52 éléments, qui sont eux-mêmes des tuples de 2 entiers. Cette liste de tuples contiendra les caractéristiques de chacune des 52 cartes. Pour chacune d'elles, il faut en effet mémoriser séparément un entier indiquant la valeur (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, les 4 dernières valeurs étant celles des valet, dame, roi et as), et un autre entier indiquant la couleur de la carte (c'est-à-dire 3,2,1,0 pour Cœur, Carreau, Trèfle et Pique).
  Dans une telle liste, l'élément (11,2) désigne donc le valet de Trèfle, et la liste terminée doit être du type :
  [(2, 0), (3,0), (3,0), (4,0), ... (12,3), (13,3),(14,3)]
- méthode **nom_carte()** : cette méthode doit renvoyer, sous la forme d'une chaîne, l'identité d'une carte quelconque dont on lui a fourni le tuple descripteur en argument. Par exemple, l'instruction : **print(jeu.nom_carte((14, 3)))** doit provoquer l'affichage de : **As de pique**
- méthode **battre()** : comme chacun sait, battre les cartes consiste à les mélanger. Cette méthode sert donc à mélanger les éléments de la liste contenant les cartes, quel qu'en soit le nombre.
- méthode **tirer()** : lorsque cette méthode est invoquée, une carte est retirée du jeu. Le tuple contenant sa valeur et sa couleur est renvoyé au programme appelant. On retire toujours la première carte de la liste. Si cette méthode est invoquée alors qu'il ne reste plus aucune carte dans la liste, il faut alors renvoyer l'objet spécial **None** au programme appelant. Exemple d'utilisation de la classe **JeuDeCartes()** :

```
jeu = JeuDeCartes() # instanciation d'un objet
jeu.battre() # mélange des cartes
for n in range(53): # tirage des 52 cartes :
    c = jeu.tirer()
    if c == None: # il ne reste plus aucune carte
        print('Terminé !') # dans la liste
    else:
        print(jeu.nom_carte(c)) # valeur et couleur de la carte
```

Exercice 3

Complément de l'exercice précédent : définir deux joueurs A et B. Instancier deux jeux de cartes (un pour chaque joueur) et les mélanger. Ensuite, à l'aide d'une boucle, tirer 52 fois une carte de chacun des deux jeux et comparer leurs valeurs. Si c'est la première des deux qui a la valeur la plus élevée, on ajoute un point au joueur A. Si la situation contraire se présente, on ajoute un point au joueur B. Si les deux valeurs sont égales, on passe au tirage suivant. Au terme de la boucle, comparer les comptes de A et B pour déterminer le gagnant.

Exercice 4

Écrivez un nouveau script qui récupère le code de l'exercice 12.2 (compte bancaire) en l'important comme un module. Définissez-y une nouvelle classe **CompteEpargne()**, dérivant de la classe **CompteBancaire()** importée, qui permette de créer des comptes d'épargne rapportant un certain intérêt au cours du temps. Pour simplifier, nous admettrons que ces intérêts sont calculés tous les mois.
Le constructeur de votre nouvelle classe devra initialiser un taux d'intérêt mensuel par défaut égal à **0,3 %**. Une méthode **changeTaux(valeur)** devra permettre de modifier ce taux à volonté.
Une méthode **capitalisation(nombreMois)** devra :

- afficher le nombre de mois et le taux d'intérêt pris en compte ;
- calculer le solde atteint en capitalisant les intérêts composés, pour le taux et le nombre de mois qui auront été choisis.

Exemple d'utilisation de la nouvelle classe :

```
>>> c1 = CompteEpargne('Duvivier', 600)
>>> c1.depot(350)
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 950 euros.
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.3 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 984.769981274 euros.
>>> c1.changeTaux(.5)
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.5 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 1045.50843891 euros.
```

Modifier le code pour que quand on récupère un solde négatif, le message suivant s'affiche : « Attention, le solde est négatif ! ». L'utilisateur ne doit pas avoir à utiliser de mutateurs/accesseurs.

Exercice 5

On veut créer des robots avec des compétences différentes. Certains sont des robots-médecins, d'autres des robots-soldats (on pourrait même combiner les deux !). Chaque robot sait se présenter, en donnant au moins son nom, et sa compétence. Chaque robot a donc un nom et un niveau de santé entre 0 et 1. On considère qu'un robot doit être soigné si son niveau de santé est inférieur à un certain seuil, qui peut être fixé globalement pour la classe Robot().
Concernant les robots-soldats, ils ont chacun une force qui correspond aux dégâts qu'il peut infliger.
Commencez à préparer un programme qui peut simuler un ensemble de robots qui se battent et/ou se soignent. A vous d'être créatif !

Cet exercice sera à rendre par mail avant le 29 septembre à 23h59, et sera noté.