TD – POO

Exercice 1 – Maillon (v. fichier) – Modifier/utiliser une classe

Suivre les indications données dans le fichier.

Exercice 2 – Cercle – Créer/utiliser une classe et la rendre débogable

Création d'une classe simple :

On souhaite créer une classe Cercle avec le contrat suivant :

- Une instance de Cercle doit avoir un rayon et un centre.
- Ces informations doivent être implantées sous la forme d'attributs « rayon », « x » et « y ».
- Un objet Cercle doit être instancié avec trois arguments, donnant dans l'ordre : la position du centre sur l'axe des abscisses, puis celle sur l'axe des ordonnées, et enfin le rayon du cercle.
- Un objet Cercle doit pouvoir donner son périmètre (sous la forme d'une méthode).
- Un objet Cercle doit pouvoir donner son aire (sous la forme d'une méthode)

1) Le 3e point ci-dessus implique de coder une méthode en particulier, dans la classe Cercle. Donner son nom et sa signature.
2) Implanter la classe Cercle.
3) Créer un Cercle centré en $(10,15)$ et de rayon de 4,3 et l'assigner à la variable $cercle$. Lancer le code puis afficher $cercle$ dans la console. On constate que l'information n'est « pas franchement » utilisable/intéressante. Pas du tout, en fait…

Déboguer du code utilisant des classes :

Pour rendre la classe débogable, on va lui ajouter une méthode particulière : $\_\_repr\_\_$ (avec 2 underscores de chaque côté). Cette méthode ne prend pas d'argument (enfin… ;-p) et renvoie une chaîne de caractères représentant les données de l'instance. Idéalement, la chaîne de caractères renvoyée doit correspondre exactement au code utilisé pour créer l'instance.

4) Ajouter la méthode $\_\_repr\_\_$ à la classe Cercle : vous aurez besoin de la méthode $format$, des chaînes de caractères (ou alors, utilisez des f-strings, si vous savez ce que c'est et que vous avez une version de python égale ou supérieure à la version 3.6).
5) Relancer le code et réafficher $cercle$ dans la console et vérifier que l'affichage est correct. Regarder également dans l'onglet variable de votre IDE et recherchez ce qui est annoncé pour la variable $cercle$. C'est mieux, non… ?
6) Toujours ***dans la console*** (PAS dans le fichier !) : muter maintenant l'instance $cercle$ pour lui donner un rayon de 22, puis réafficher l'instance dans la console. Vérifiez que le rayon est bien à jour dans l'affichage.

Augmenter la classe Cercle :

On veut maintenant ajouter une méthode $intersect$ à la classe, qui va prendre en argument une autre instance de la classe Cercle et qui renverra un booléen indiquant si le cercle passé en argument coupe celui représenté par l'instance en cours ou pas.

7) Si on considère deux cercles représentés par les données R_1, x_1, y_1 et R_2, x_2, y_2, comment sait-on si les deux cercles se coupent ou pas ? Donner la formule.

8) Implanter la méthode $intersect$ en convertissant la logique précédente. Cette méthode doit au final être utilisée de la façon suivant :

   $c1 = Cercle(\ x_1, y_1, r_1)$

   $c2 = Cercle(\ x_2, y_2, r_2)$

   $c1.instersect(c2)$ $\# \rightarrow\ bool$

9) Définir une méthode $filtrer$, qui prend en argument une liste de cercles et renvoie une nouvelle liste des cercles qui ont une intersection avec l'instance en cours : $cercle.filtrer(list\_of\_Cercles)$. L'utiliser ensuite pour trouver tous les cercles coupant l'instance $cercle$ définie au début, parmi les cercles de la liste lst définis dans le tout dernier test en bas du fichier (nota : l'ordre des cercles dans la liste ne doit pas changer).

Exercice 3 – Pneu (pas de fichier associé)

On souhaite modéliser des pneus… Un pneu est défini par différentes caractéristiques :

- Son nom de gamme
- Le type de gomme
- Sa pression interne
- La pression maximale acceptable
- La distance totale parcourue par le pneu

1) Quels sont les arguments qui seront passés au constructeur de cette classe ? (une instance de Pneu est créée lorsqu'un pneu est monté sur gente. Il ne sera alors plus démonté jusqu'à ce qu'il soit ~~jeté~~ recyclé).
2) Écrire (sur feuille) le code permettant de déclarer la classe et de créer une instance de cette classe, avec tous ses attributs.

Un pneu peut être gonflé ou dégonflé. Ces opérations sont faites via les méthodes $pneu.gonfle(p)$ et $pneu.degonfle(p)$ où les arguments p sont des valeurs positives de variations de pression (à ajouter ou à enlever, donc). Dans tous les cas, la pression interne ne doit jamais devenir négative, ou passer au-dessus de la pression maximale acceptable (on limitera simplement les valeurs de pression à ces limites, sans lever aucune exception, si cela arrive).

3) Écrire le code permettant de définir ces deux méthodes dans la classe Pneu, en s'assurant que les conditions ci-dessus seront respectées.

En python, les propriétés des objets sont accessibles « depuis l'extérieur » des objets. C'est-à-dire que si une instance $pneu$ est définie, l'utilisateur du code peut toujours accéder à la valeur ou la modifier :

$$penu.pression += 2.6$$

4) Quel est le risque ici ? Quel concept est en jeu dans cet exercice ? Qu'implique-t-il ?

Exercice 4 - Fractions

La classe $Fraction$ représente les nombres fractionnaires en stockant deux attributs, num et $denom$, correspondant respectivement au numérateur et au dénominateur de la faction. L'intérêt de ce type de classe est qu'elle permet de représenter des nombres irrationnels, ou des nombres décimaux très petits ou très grands sans perte de précision, contrairement au type $float$ (qui a une précision de 64 bits moins le nombre de bits utilisés pour encode l'exposant et le bit de signe).

1) Dans le fichier python, créer la classe $Fraction$, avec uniquement son constructeur. Une instance de $Fraction$ doit avoir deux attributs, num et $denom$, et $Fraction(x, y)$ permet de créer la fraction représentant x/y.
   ATTENTION : concernant la gestion des signes, on veut que, si la fraction est négative, seul le numérateur porte le signe, et si la fraction est positive, le numérateur et le dénominateur doivent toujours être positifs. Dresser un tableau avec tous les signes possibles pour les arguments et les résultats attendus, et en déduire un « algorithme » pour mettre en œuvre la spécification voulue concernant les signes.

2) Ajouter la méthode $\_repr\_$ à la classe, de manière à pouvoir déboguer votre code plus facilement.

3) On souhaite maintenant pourvoir créer des instances de $Fraction$ représentant des entiers, comme $13/1$, en n'utilisant qu'un seul argument pour le constructeur : $Fraction(13)$. Modifier le code en conséquence.

Cette classe ne sert pour le moment à rien… Puisqu'on ne peut pour l'instant pas faire de calculs avec. On va donc implanter maintenant les quatre opérations classiques, avec les noms de méthodes suivants : add, sub, mul et div. La signature de ces ***méthodes*** est toujours :

$$frac.operation(autre: Fraction) \rightarrow Fraction$$

Où $frac$ est l'instance de Fraction représentant le terme de gauche dans le calcul. Par exemple :

Soient $f_1 = Fraction(1,4)$ et $f_2 = Fraction(7,5)$, la soustraction des deux fractions s'écrira :

$$\frac{1}{4} - \frac{7}{5} \rightarrow \cdots \qquad \Rightarrow \qquad f_1.sub(f_2) \rightarrow Fraction(\ldots, \ldots)$$

Chaque méthode devra renvoyer une nouvelle instance de Fraction, sans modifier les deux instances utilisées pour faire le calcul.

4) Soient deux fractions a/b et c/d, écrire les formules littérales donnant le résultat des quatre opérations possibles en prenant toujours a/b pour la terme de gauche dans le calcul.

5) Implanter les différentes add, sub, mul et div, en gardant bien en tête que chaque méthode doit renvoyer une nouvelle instance de la classe $Fraction$ représentant le résultat du calcul.

6) Définir maintenant la variable $result$ qui devra contenir le résultat du calcul suivant (utiliser la classe pour faire les calculs…) :

$$result = \left(\frac{1}{4} + \frac{15}{7}\right) * \frac{5}{3} - 4$$

Raffinements de l'implantation :

7) Quel est le résultat de $\frac{5}{4} + \frac{3}{2}$, mathématiquement ? Quel est le résultat en utilisant la classe Fraction ? Conclusion ?

On voit qu'on peut ajouter une logique de simplification des fractions, en passant par le calcul de plus grand diviseur commun pour deux nombres positifs, dont l'algorithme est le suivant :

- $pgdc(a, b)$ vaut a si b est nul
- $pgdc(a, b)$ vaut $pgdc(b, a\%b)$ si b est non nul
- Si a ou b sont négatifs, on applique le même raisonnement, mais sur leur valeur absolue.

8) Implanter en dehors de votre classe la fonction récursive pgdc.

9) Cette fonction doit permettre de corriger les résultats de toutes les méthodes impliquant des opérations mathématiques. À quel endroit doit-elle être utilisée et pourquoi ? Faire le nécessaire.

10) Il y a un dernier test $test\_big$, qui échoue probablement à ce stade. La classe Fraction fonctionne déjà « correctement » (sauf dans certains cas comme pour ces tests), mais vous pouvez essayer de comprendre d'où vient le problème et chercher comment le résoudre… (*hint : il y a en tout et pour tout deux caractères à ajouter quelque-part dans le code… normalement…)*.