Algo & Prog, avec Python
(L1-Sciences)
TP n° 10, Automne 2016

Classes et Objets

Exercice 10.1 Mon petit frère doit passer le bac à la fin de l'année et il veut que je l'aide à réviser la résolution des équations du deuxième degré. Comme je suis convaincu qu'il vaut mieux faire les calculs à l'ordinateur que les faire de tête, je décide de réaliser un programme en Python. Proposez une classe `Poly2` qui modélise un polynôme du second degré de la forme ax^2+bx+c (rappel : en maths, la lettre « x » représente une *indéterminée*). On souhaite disposer des méthodes suivantes :

`valeur`	# retourne la valeur en un point `x` du polynôme
`delta`	# retourne le discriminant $\Delta = b^2-4ac$ du polynôme
`racines`	# retourne le couple (r_1,r_2) des racines réelles avec $r_1 \le r_2$, ou `False`
`__repr__`	# retourne une représentation sous la forme d'une chaîne `'5x²-x+3'` [sous Unicode UTF-8 le caractère ² en exposant a pour numéro 178].

N.B. Dans le cas où $\Delta < 0$, la méthode `racines` retournera `False`.

Exercice 10.2 Calculez les racines des polynômes $2x^2-3x-2$, x^2-4x-4, x^2+1.

Exercice 10.3 *Prolongement de l'exercice 1 aux racines complexes*. Vous avez de la chance, il existe une classe `complex` d'emblée intégrée à Python 3, donc sans `import`.

a) Cherchez sur le Web (Google : « nombres complexes en python 3 ») comment définir en Python avec et sans le mot `complex` les nombres complexes z_1=`3-2i`, z_2=`5+i` et z_3=`i`. Calculez l'addition z_1+z_2, le produit z_1z_2 et l'inverse `1/`z_3. Vérifiez si z_3 est bien la racine carrée de `-1`…

b) Reprogrammez la méthode `racines` de l'exercice 1 qui retournera dans tous les cas un couple (z_1,z_2) formé de deux racines complexes.

c) Calculez les racines des polynômes $2x^2-3x-2$, x^2+x+1, x^2+1

Exercice 10.4 Avec les bonnes résolutions d'un début d'année, on veut se préparer aux résultats des examens qui viennent toujours trop vite. Pour cela, vous allez écrire une classe permettant le calcul de la moyenne. Nous utiliserons des nombres approchés et nous stockerons les notes dans un attribut d'instance `L`. L'utilisateur débutera un calcul de moyenne en créant un objet de type `Moyenne` avec une liste de notes :

```
anglais = Moyenne([12,8,10])
```

Il pourra à tout moment demander la moyenne des notes entrées, et il pourra saisir une nouvelle note avec la méthode `saisir_note()` qui utilisera `input('Entrez une note')` pour lire une note au clavier. Enfin, une méthode `min_max` permettra d'obtenir le couple *(minimum,maximum)*. Programmez la classe `Moyenne`.

Algo & Prog, avec Python
(L1-Sciences)
TD n° 10, Automne 2016

Exercice 10.1 a) Dites précisément ce qu'on entend par *attribut d'instance*, *méthode d'instance*, *attribut de classe*. Qu'appelle-t-on *l'état* d'un objet ?
b) Dans le cours page 14, pourrait-on faire de `cercles` une variable initialisée à l'extérieur de la classe `Cercle` ?

Exercice 10.2 Programmons une classe `Vect2d` dont les objets modéliseront des vecteurs à deux composantes réelles. Lorsque le mathématicien dit : « Soit le vecteur $\vec{u}(3,-2)$ », le programmeur Python dira : « `u = Vect2d(3,-2)` ».

a) Programmez le *constructeur*, chargé d'initialiser les champs `x` et `y` de l'instance courante, qui se nomme `self`. Par défaut, `x` et `y` seront mis à 0.
b) Programmez une méthode chargée de produire une chaîne de caractères représentant l'état de l'objet. Cette fonction porte en Python un nom particulier !
c) Si je veux additionner deux vecteurs, j'ai deux solutions. Ou bien je programme une fonction mathématique `add(v1,v2)` à l'extérieur de la classe[1]. Ou bien – dans une optique de pure programmation par objets – je programme `add` comme méthode d'instance à l'intérieur de la classe. Adoptez cette seconde solution, pour laquelle `add` devient un *message* envoyé à un vecteur. Le résultat de l'addition sera un nouveau vecteur, aucune mutation !
d) Programmez une *méthode* `mul_ext` réalisant la multiplication $k\vec{u}$ d'un réel `k` par un vecteur $\vec{u}$. Le résultat sera un nouveau vecteur, aucune mutation.
e) Programmez une *méthode* `zoom` demandant à un vecteur de bien vouloir se multiplier par `k` et d'être ainsi définitivement modifié ! Faites bien la différence avec `mul_ext`.
f) Programmez une méthode `prodscal` demandant à un vecteur de retourner son produit scalaire avec un autre vecteur.
g) En déduire une méthode `norme` demandant à un vecteur de retourner sa longueur.
h) Programmez à l'extérieur de la classe une fonction `add(v1,v2)` retournant la somme vectorielle $\vec{v}_1 + \vec{v}_2$.

N.B. Est-il possible d'écrire u + v si u et v sont deux vecteurs ? En principe non, mais en Python, des ***méthodes spéciales*** *sont cachées sous les opérateurs. Si votre méthode add se nomme __add__, cela devient possible !*

Exercice 10.3 On souhaite modéliser un *climatiseur* par la classe `Climatiseur`. Un climatiseur permet d'obtenir une température souhaitée en degrés Celsius. Lorsque je commande un climatiseur à l'usine, je spécifie la plage de températures de fonctionnement `[tmin,tmax]` : il ne descendra jamais en-dessous de `tmin` et ne dépassera jamais `tmax`. Je peux aussi spécifier la température souhaitée au démarrage `tinit`. Si je ne la demande pas, l'usine le règlera à 20° par défaut. Un climatiseur est muni de deux commandes pour `augmenter` ou `diminuer` la température souhaitée de 1°C (en restant dans la plage de fonctionnement). Il a enfin un bouton `afficher` qui affiche avec `print` la température à la fois en °C et en °F. On rappelle que : `°C = (°F - 32) *` $\frac{9}{5}$

[1] Vous verrez plus tard la possibilité de programmer `add` comme *méthode de classe*…