https://www.linkedin.com/pulse/passer-de-la-seconde-%C3%A0-vitesse-sup%C3%A9rieure-en-python-olivier-le-goa%C3%ABr/

Passer de la seconde à la vitesse supérieure en Python

Published on 2019 M04 2

Olivier Le Goaër

Maître de Conférences en Informatique

En 2017, Python a été propulsé par le ministère comme langage quasi-officiel pour l'apprentissage de la programmation dès la classe de seconde. Ce billet illustre comment un exercice de Math considéré comme élémentaire peut donner matière à des améliorations non triviales permises par le langage Python. C'est aussi une façon de montrer une continuation entre l'enseignement de Python dans le secondaire et son enseignement dans le supérieur.

Cas d'école : calculer l'équation d'une droite

Il existe un exercice incontournable de programmation Python prévu en classe de seconde, et qui peut tout à fait se décliner à différents niveaux de difficultés. En voici l'énoncé :

> Soient deux points A et B. On souhaite automatiser les calculs permettant d'obtenir l'équation réduite de la droite (AB). Élaborer une démarche.

Il s'agit d'un exercice faisant appel aux connaissances des élèves sur les fonctions affines. Guidée par l'enseignant.e, la bonne démarche inclue la création d'une fonction Python. La solution-type pour cet exercice est la suivante :

```
def eq_droite(xA,yA,xB,yB):
    if yA==yB:
        print("L'équation réduite de (AB) est : y= ", yA)
    else:
        a=(yB-yA)/(xB-xA)
        b=yA-a*xA
        print("L'équation réduite de (AB) est : y= ",a, "x+ ",b)
```

La façon de traiter cet exercice a récemment fait l'objet d'un débat entre des enseignants de mathématiques et un inspecteur pédagogique au sein de MathémaTICE. Au cœur des échanges : le cas où les points A et B sont identiques (car pas de droite), la place des *print* et plus généralement le "retour" de ce genre de fonction.

En réalité, la solution efficiente à cet exercice se trouve dans des concepts éprouvés de génie logiciel et implantés dans le langage Python, mais clairement hors-programme au Lycée.

On passe la seconde !

Python possède des mécanismes natifs [1] pour solutionner l'exercice de façon élégante. En voici trois :

Structure de donnée

Littéralement, on calcule une équation à partir de deux points, et non pas à partir de coordonnées. Coller au plus près à l'énoncé mathématique revient à créer une structure de données pour modéliser les points. Un choix raisonnable est de considérer qu'un point est un 2-uplet de nombres, grâce au type *Tuple* de Python. On passe ainsi de 4 paramètres à seulement deux, ce qui implique au passage moins de risque d'erreurs.

Assertion

Pourquoi calculer une équation de droite alors que c'est impossible ? L'algorithme n'a de sens que si les données qui lui sont fournies en ont un. C'est la base d'un contrat tacite passé entre le créateur du programme et celui qui va l'utiliser, et qui peut être rendu explicite en Python à l'aide du mot clé *assert*. Les assertions sont des expressions logiques qui, si elles sont évaluées à Faux, empêchent l'exécution du programme en levant une *AssertionError* récupérable si besoin avec *try/except*.

Fermeture

Admettons-le, afficher une équation à l'écran présente une utilité très limitée en situation réelle. Il serait beaucoup plus pertinent de renvoyer l'équation calculée, en tant que fonction Python à part entière, pour pouvoir l'utiliser à son tour. Ceci est possible grâce aux fermetures lexicales (*Closures* en anglais). Ce sont des fonctions définies à l'intérieur d'une fonction hôte, ayant accès à son contexte, et pouvant être renvoyées par cette dernière.

Voici la nouvelle fonction :

```
def eq_droite(pointA, pointB):
    xA, yA = pointA
    xB, yB = pointB
    assert not (pointA==pointB), "Same points not allowed"
    assert not (xA==xB), "Same x-axis not allowed"

    def constantFct(x):
        return yA

    def linearFct(x):
        a=(yB-yA)/(xB-xA)
        b=yA-a*xA
        return a*x + b

    return (constantFct if yA==yB else linearFct)
```

Et voici sa nouvelle utilisation :

```
A = (4,2)
B = (5,-6)
f = eq_droite(A,B)  #renvoie la fonction y=-8x+34
print(f(-89))       #test avec x=-89
```

Toujours plus vite

Ce petit exercice d'apparence inoffensif nous a amené en un rien de temps à introduire des concepts

plus fins du langage Python. Il peut même encore donner du grain à moudre aux enseignants et aux étudiants :

- Le *Tuple* n'est pas la structure de donnée miracle pour modéliser les points, car la confusion abscisse/ordonnée reste possible. En cela, le type dictionnaire *Dict* est intéressant car il associe des clés aux valeurs, mais on peut aussi se tromper sur les clés... La solution infaillible est la création d'une classe 'Point', possible uniquement dans des langages orientés objet comme Python.
- Dans un langage sans typage explicite comme Python, réduire le nombre de paramètres de 4 valeurs basiques à deux objets complexes peut paradoxalement entraîner une hésitation dans l'esprit de l'utilisateur sur ce qu'on attend de lui. Elle peut être atténuée par des indices de types (*Type Hints*) déposés au niveau de la signature de la fonction.
- Je suis tombé sur une solution à l'exercice où l'enseignant.e proposait de renvoyer l'équation $x=c$ dans le cas où les abscisses sont identiques, là où j'ai considéré cette situation comme une violation pure et simple du contrat. Preuve que l'intention de la fonction n'est pas forcément communément partagée. La solution ? Avoir recours à de la documentation intégrée (*docstrings*) de sorte que la commande *help(eq_droite)* annonce clairement la couleur.

[1] Olivier Le Goaër. Cours Python (89 diapos) :
https://olegoaer.developpez.com/cours/python/Python.pdf

Olivier Le Goaër

Maître de Conférences en Infor