Des questions sur Python

publié le 23/06/2019 - mis à jour le 24/06/2019

Descriptif :

Une liste de questions et de réponses sur Python dans le cadre des nouveaux programmes de Seconde et Première.

Sommaire :

Suite aux différents stages qui ont eu lieu dans l'académie de Poitiers sur les nouveaux programmes de lycée, certaines questions sont revenues régulièrement au sujet de Python. Les voici avec les réponses :

● Quelle différence existe entre = et == ?

= sert à affecter la valeur à droite du signe = à la variable dont le nom est à gauche.
== est un opérateur qui permet de vérifier si deux variables ont la même valeur ou pas.

```
x = 5
```

```
y = 6
if x == y :
  print("Les valeurs sont identiques")
else :
  print("Les valeurs sont différentes")
```

● Pourquoi Python fait des erreurs de calculs ?

Il n'y a aucune erreur de calcul mais certains résultats peuvent sembler pour le moins surprenants.
Avant de voir l'explication, essayons de deviner ce qui va être affiché avec le code suivant :

```
if 3/9 == 0.003/0.009 :
  print("Ce sont les mêmes valeurs.")
else :
  print("Ce sont des valeurs différentes.")
```

La réponse est **Ce sont des valeurs différentes** alors que mathématiquement ce sont bien deux valeurs identiques.
Cela peut poser problème en particulier dans l'activité de Première "Déterminer la composition de l'état final d'un système siège d'une transformation chimique totale à l'aide d'un langage de programmation"

Pour comprendre, affichons les deux valeurs suivantes :

```
print(3/9)
print(0.003/0.009)
```

A l'écran :

0.3333333333333333
0.3333333333333337

Effectivement, ce sont bien deux valeurs différentes. **Mais pourquoi ce 7 à la fin ?**

Tous les nombres décimaux sont stockés en binaire. Par exemple, puisque : 13 = $\mathbf{1}x2^3+\mathbf{1}x2^2+\mathbf{0}x2^1+\mathbf{1}x2^0$, 13 s'écrit en binaire : 1101
Pour les nombres flottants, c'est le même principe mais avec des puissances négatives de 2, autrement dit $(1/2)^n$

Par exemple $0.625=\mathbf{1}x(1/2)^1+\mathbf{0}x(1/2)^2+\mathbf{1}x(1/2)^3$ donc 0.625 s'écrit 0.101 en binaire. Malheureusement, un nombre décimal (c'est à dire avec un nombre de chiffre fini après la virgule) peut avoir un développement binaire infini. Par exemple 0.1 en décimal devient en binaire une valeur avec un nombre infini de chiffres après la virgule :
$0.1=\mathbf{0}x(1/2)^1+\mathbf{0}x(1/2)^2+\mathbf{0}x(1/2)^3+\mathbf{1}x(1/2)^4+\mathbf{1}x(1/2)^5+\mathbf{0}x(1/2)^6+\mathbf{0}x(1/2)^7+\mathbf{1}x(1/2)^8+\mathbf{1}x(1/2)^9+\mathbf{0}x(1/2)^{10}+\mathbf{0}x(1/2)^{11}+\mathbf{1}x(1/2)^{12}+....$
0.1 s'écrit donc en binaire 0.000110011001...

Par ailleurs un nombre à virgule dans Python, un flottant, se code sur 53 bits. La valeur 0.1 devient donc une valeur arrondie en binaire. Si on convertit à nouveau en décimal, on obtient :
0.1000000000000000055511151231257827021181583404541015625

En conséquence, si on tape

```
print(0.1)
print(0.1+0.1+0.1)
```

A l'écran :

0.1
0.30000000000000004

En effet, Python 3 affiche pour 0.1, une valeur à nouveau arrondie lors de la conversion *binaire -> décimal* qui permet de retrouver 0.1 mais l'opération 0.1+0.1+0.1 amplifie l'écart et ce n'est pas 0.3 qui apparait.

Comment résoudre ce problème ?
La fonction round affiche une valeur arrondie lorsque les calculs sont terminés : **round(valeur, nombre de chiffre après la virgule)**. Dans le code ci-dessous, print affiche la valeur en arrondissant au quatrième chiffre après la virgule :

```
somme = 0.1+0.1+0.1
print(round(somme,4))
```

A l'écran :

0.3

En effet, les 0 après le 3 ne sont pas affichés.
Pour information, si l'on fait :

```
somme = round(0.1)+round(0.1)+round(0.1)
print(somme)
```

A l'écran :

0.30000000000000004

On a retrouvé le même problème car l'arrondi n'a pas été fait sur le résultat final.

Mais attention, pour les mêmes raisons, round renvoie des arrondis qui peuvent sembler étrange :

```
print (round(2.675,2))
print (round(2.685,2))
```

A l'écran :

2.67
2.69

Donc la fonction round() n'est pas judicieuse pour vérifier des égalités entre deux valeurs.

Si le programme que l'on tape, exige de ne pas rencontrer ces problèmes, Python propose plusieurs solutions mais aucune n'est transparente pour l'élève. On peut par exemple faire appel au module décimal de la bibliothèque Python :

```
import decimal
valeur = decimal.Decimal('0.3')
somme = valeur+valeur+valeur
print(somme)
```

A l'écran :

0.9

La documentation de Python précise :

> *Le module decimal « est basé sur un modèle en virgule flottante conçu pour les humains, qui suit ce principe directeur : l'ordinateur doit fournir un modèle de calcul qui fonctionne de la même manière que le calcul qu'on apprend à l'école »*

● Comment choisir le nombre de chiffres après la virgule quand on affiche une valeur ?

```
valeurA = 0.532
valeurB = 0.502
print (round(valeurA,2))
print (round(valeurB,2))
```

A l'écran :

0.53
0.5

En effet, pour 0.5, le dernier 0 est considéré comme "inutile" donc il n'est pas affiché.
Pour obtenir 0.50, il faut utiliser un formatage spécial dans le print :

```
valeur = 0.502
print ("La valeur arrondie au deuxième chiffre après la virgule est %.2f"%(valeur))
```

A l'écran :

La valeur arrondie au deuxième chiffre après la virgule est 0.50

La variable *valeur* a été affichée en tant que flottant(f) avec 2 chiffres après la virgule(2f). Remarquez la présence des deux symboles "pourcentage". Pour information, ce formatage est apparu dans un sujet 0 des E3C pour les Premières
.

● Comment afficher une valeur en notation scientifique ?

Il faut formater l'affichage ainsi :

```
valeur = 138750
print ("La valeur est %E"%(valeur))
```

A l'écran :

La valeur est 1.387500E+05

On peut aussi imposer le nombre de chiffre après la virgule :

```
valeur = 138750
print ("La valeur est %.2E"%(valeur))
```

A l'écran :

La valeur est 1.39E+05

● Peut-on écrire x = x+0.3 ?

En mathématique, c'est une équation sans solution. Mais en langage informatique le signe = permet d'affecter la valeur

de droite à la variable qui est à gauche. Donc dans un premier temps, x+0.3 est évalué puis x reçoit cette nouvelle valeur. Si x valait 2.6, x vaut à présent 2.9 .
On va, par exemple, utiliser cette affectation quand on veut incrémenter une variable dans une boucle avec un pas en particulier.

● Pourquoi utiliser quiver et pas arrow pour dessiner une flèche ?

On peut très bien utiliser **arrow**. C'est une fonction simple à utiliser et son nom est évocateur pour les élèves. Mais elle présente un inconvénient : dans le cas d'un repère qui n'est pas orthonormé, la flèche a une apparence déformée. Pour qu'elle ait l'apparence attendue par les élèves, il faut calculer une mise à l'échelle qui n'est pas au cœur du programme de Physique Chimie. **quiver** est une fonction qui permet beaucoup plus de possibilités que **arrow**. La plupart sont inutiles pour les activités de Seconde et de Première. Mais avec les trois derniers paramètres ci-dessous, la flèche n'est jamais déformée :

```
plt.quiver(x0,y0,deltax,deltay,  angles='xy', scale=1, scale_units='xy')
```

● Comment faire une flèche colorée avec quiver ?

Il faut ajouter le paramètre *color* et utiliser les lettres désignant les couleurs comme dans plt.plot :

```
plt.quiver(x0,y0,deltax,deltay,  angles='xy', scale=1, scale_units='xy', color='r')
```

● Quels sont les mots réservés par Python qu'il ne faut pas employer pour le nom d'une variable ?

Pour nommer une variable, il ne faut pas utiliser les mots qui sont déjà utilisés par Python. Vous pouvez afficher la liste *"interdite"* grâce au code suivant :

```
import keyword
import builtins
print(dir(builtins),keyword.kwlist)
```

● Comment supprimer ou insérer un élément dans une liste ?

Il y a eu plusieurs questions portant sur la manipulation des listes.

```
uneliste.pop() # supprime le dernier élément de uneliste
uneliste.insert(i,element)  # insère element dans uneliste à l'indice i
```

Normalement, ce n'est pas nécessaire pour les activités de Seconde et Première mais si vous le souhaitez, vous trouverez toutes les possibilités sur cette documentation de Python.

● Propositions de collègues

Par ailleurs, des collègues ont proposé des techniques qui n'apparaissent pas dans le cours. Merci à eux, les voici :

- La fonction **plt.grid(color='r', linestyle='-', linewidth=2)** permet l'affichage d'une grille avec matplotlib.pyplot
- **plt.axis("equal")** permet d'avoir un repère orthonormé mais il y a deux inconvénients. Bien sûr les courbes peuvent avoir un aspect "écrasé". Par ailleurs, le calcul automatique de la longueur des axes ne prend pas en compte les dimensions des flèches tracées avec quiver ou arrow pour qu'elles apparaissent dans l'image.

Si vous voulez partager des informations, des astuces ou bien des remarques, n'hésitez pas à les envoyer.