Devoir surveillé n° 01 - Remarques

Barème.

Partie machine, chaque question sur 2 points, total sur 24 points ; partie papier, 2 points pour la première question, 3 pour les autres, total sur 23 points ; total sur 47 points, ramené sur 20. points.

Statistiques descriptives.

	Papier	Machine	Note finale
Note maximale	24	$18,5$	18
Note minimale	0	$0,5$	2
Moyenne	$\approx 14,21$	$\approx 10,5$	$\approx 10,49$
Écart-type	$\approx 5,26$	$\approx 4,80$	$\approx 3,88$

Remarques sur la partie papier.

Remarques générales.

— Pensez à écrire systématiquement une docstring en préambule à vos fonctions python. Une docstring commence et finit par `"""`, et ce symbole permettant les retours à la ligne, il n'y a pas d'autre caractère à utiliser pour délimiter les commentaires. Attention : la docstring doit être indentée, comme tout le corps de la fonction.
Une docstring doit contenir deux éléments : une description de la fonction, et une description des arguments (en particulier, leur type).

— Les noms de variables et de fonctions ne doivent contenir aucun espace ni aucun accent. Évitez également le caractère `-`, qui peut être interpréter comme une soustraction. Préférez-lui le caractère `_`.

— Même à l'écrit, les caractères utilisés et la disposition du texte doivent être absolument identiques à ce que vous taperiez sur une machine. Même si cela n'est pas rédhibitoire contrairement à un script python éxécuté sur une machine, l'omission des `:`, les `def`, `return` ou autres mots-clés commençant par une majuscule (ce qui ne devrait pas être le cas) peuvent vous faire perdre des points et ne laissent pas préjuger d'une grande aisance en python. Une indentation qui n'apparaît pas clairement sera par contre sanctionnée car différentes indentations mènent à des codes ayant des sens différents. Soyez donc aussi rigoureux sur ces points que vous ne le seriez devant une machine.

— Les notions d'invariant et de variant sont encore mal comprises. Il n'y a de variant que pour une boucle `while`. Le variant est un objet python, pas un énoncé. Ce doit être un objet qui intervient dans le test de la boucle `while` en question. Et c'est le fait que sa valeur change qui permet à la boucle de se terminer. Un variant ne sert donc qu'à démontrer la terminaison d'une boucle `while`.
Un invariant est quant à lui un énoncé, qui doit dépendre du tour de boucle d'une boucle `for` ou `while`. C'est une hypothèse de récurrence : un invariant ne dépendant pas du tour de boucle est comme une hypothèse de récurrence ne dépendant pas de n (dans ce cas ce n'est pas une récurrence !). Et c'est le fait que cet énoncé est vrai à la fin du dernier tour de boucle, qui assure que la fonction renvoie le résultat voulu. Un invariant qui ne permet pas directement de montrer cela n'est pas nécessairement faux, mais il n'a aucun intérêt.

— Écrivez les `import truc` en dehors des fonctions.
— Un booléen, c'est `True` ou `False` et rien d'autre, et écrit comme ça et pas autrement. Si on vous demande une fonction qui retourne un booléen, ça doit se finir par `return True` ou `return False`, et rien d'autre. En particulier, `print blabla` ne renvoie rien du tout !
— Utiliser des fonctions toutes faites comme `sum` ou surtout `factorial` dans des programmes aussi simple et considéré comme une petite tricherie.
— La commande la plus efficace pour ajouter un élément `e` dans un tableau `t` est `t.append(e)`. La commande `t = t + [e]` impose une recopie totale du tableau `t`, ce qui prend beaucoup plus de temps. Dans le même ordre d'idée, pour concaténer un tableaux `u` à la fin d'un tableau `u`, on utilisera `t += u` et non `t = t+u`, pour les mêmes raisons.

Au fil des questions.

Q1 Une majorité d'élèves à confondu disque et carré, c'est assez inquiétant. Si $a = (x, y)$ et $c = (z, t)$, dire que $|x - z| \leq r$ et $|y - t| \leq r$, c'est dire que a est dans le carré de centre a, et de côtés horizontaux et verticaux de longueur $2c$. Dire que a est dans le disque fermé de centre c et de rayon r, c'est : $(x - z)^2 + (y - t)^2 \leq r^2$ (ce qui d'ailleurs ne nécessitait pas l'utilisation de `sqrt`, fonction que d'ailleurs il faut importer avant d'utiliser).
En maths, si l'on connaît un couple a, on peut rédiger : « posons $a = (x, y)$ ». En python, on ne peut pas. La variable à gauche d'un = est le nom d'une variable que l'on crée, en fonction des varaibles qui se trouvent à droite. Écrire `a=(x,y)` amène à une erreur : `x,y` inconnus. Il faut écrire `(x,y)=a`.
a et c sont des couples, donc $a - c$ n'a pas de sens en python.

Q2 `Python parfait` n'était pas le nom attendu. Il s'agissait d'une fonction écrite en Python, et qui s'appellerait `parfait`. Une seule boucle `for` suffisait : on pouvait détecter les diviseurs et les additionner directement. Dcéomposer cela en deux boucles rend le temps de calcul deux fois plus long et utilise plus de varaibles et de mémoire.
Pour savoir si a divise b, on utilise `a% b == 0`, et surtout pas des tests bancales comme `a/b == int(a/b)` ou `a//b == a/b` ou je ne sais quoi d'autre.
`a/b` est toujours un flottant : `4/2 = 2.0`.

Q3 Pour calculer $(n + 1)!$ après avoir calculé $n!$, surtout, surtout, on stocke le calcul de $n!$ dans une variable, et on le multiplie par $n + 1$. Surtout, surtout, on ne recommence pas tout depuis le départ. Calculer toutes les factorielles de 1! à 6! demande ainsi 5 multiplications, et non $1 + 2 + 3 + 4 + 5 = 10$. Et plus n croît, pire c'est : $\frac{n(n+1)}{2}$ multiplications au lieu de n. Il ne fallait donc surtout pas utiliser une fonction calculant $n!$, mais incluire ce calcul dans une boucle `for`.