Corrigé Sujet 2 épreuve pratique NSI

Michel Billaud (michel.billaud@laposte.net)

31 janvier 2022

Table des matières

1 Licence

2 Le sujet

Se trouve sur la page https://eduscol.education.fr/2661/banque-des-epreuves-pratiques-de-specialite-nsi, dans https://eduscol.education.fr/document/33181/download

3 Exercice 1 : moyenne avec coefficient

Sujet résumé : écrire une fonction `moyenne` qui a comme paramètre une liste de couples (note, coefficient), et retourne la moyenne pondérée. Exemple,

`moyenne([(15,2),(9,1),(12,3)])` s'évalue en `12.5`, parce que

$\frac{15\times2+9\times1+12\times3}{2+1+3} = 12.5$

3.1 Résolution

La formule ci-dessus met en évidence une fraction avec un numérateur et un dénominateur qui sont des sommes.

Chaque couple contribue

- au numérateur, par le produit $note \times coefficient$,
- au dénominateur, par le $coefficient$.

Il s'agit donc de calculer deux sommes, et d'en retourner le quotient.

3.2 Version simple : en un seule boucle

```
def moyenne(liste):
    somme_produits = 0
    somme_coefficients = 0
    for (note, coefficient) in liste:
        somme_produits     += note * coefficient
        somme_coefficients += coefficient
    return somme_produits / somme_coefficients
```

Ceci suppose connus

1. la forme `for (note, coefficient) in liste:`, qui sinon peut-être écrite

   ```
   for couple in liste:
      note = couple[0]
      coefficient = couple[1]
   ```

2. L'augmentation d'une variable `somme_produits += note * coefficient` qui peut se réduire à une affectation

   ```
   somme_produits = somme_produits + note * coefficient
   ```

plus lourde et moins lisible, parce qu'elle n'exprime pas *directement* l'idée "ajouter telle quantité à une variable".

3.3 Variante, avec les listes en intension

```
def moyenne(liste):
    somme_produits     = sum(note * coefficient for (note, coefficient) in liste)
    somme_coefficients = sum(coefficient        for (note, coefficient) in liste)
    return somme_produits / somme_coefficients
```

4 Exercice 2 : Triangle de Pascal

```
def pascal(n):
    C= [[1]]
```

```
    for k in range(1,...):                          #1
        Ck = [...]                                  #2
        for i in range(1,k):
            Ck.append(C[...][i-1]+C[...][...] )     #3
        Ck.append(...)                              #4
        C.append(Ck)
    return C
```

Le texte donne (lourdement) toutes les indications nécessaires

- La variable `Ck` (contient) à l'étape numéro `k`, la k-ième ligne du tableau.
- chaque ligne `commence` et se `termine` par le `nombre 1`.
- **Par ailleurs**, la valeur (…) s'obtient en ajoutant les valeurs des deux cases situées juste au-dessus.

4.1 Résolution

En regardant les exemples, on voit que `pascal(4)` retourne une liste de 5 sous-listes (lignes). En général, le tableau `pascal(n)` contient `n+1` lignes, parce que les indices des coefficients vont de 0 à `n`.

On peut en inférer, pour le premier "trou" :

- puisqu'il faut produire un tableau de `n+1` lignes, la boucle `for` doit être exécutée `n` fois, la première ligne ayant été construite avant cela par la première instruction `C = [[1]]`
- donc `for k in range(1, n + 1)): #1`, de sorte que `k` parcoure les valeurs de 1 à `n` (attention à `range`).

Les trous 2 et 4 concernent les extrémités de la ligne, dans lesquels on place des `1`

```
Ck = [[1]]      #2
Ck.append(1)
```

Quand au trou numéro 3, c'est la traduction de "somme des deux cases au dessus".

- on est ligne `k`, donc elles sont sur la ligne `k-1`
- elles ont les indices `i` et `i-1`, ce qui se voit mieux si on dessine les coefficients dans un triangle rectangle

```
k  j| 0 1 2 3 4
---+-----------
0   | 1
1   | 1 1
2   | 1 2 1
3   | 1 3 3 1
4   | 1 4 6 4 1
```

4.2 Récapitulation

```
def pascal(n):
    C = [[1]]
```

```
for k in range(1, n + 1):                          #1
    Ck = [1]                                       #2
    for i in range(1, k):
        Ck.append(C[k-1][i-1] + C[k-1][i] )        #3
    Ck.append(1)                                   #4
    C.append(Ck)
return C
```