Corrigé Sujet 6 épreuve pratique NSI

Michel Billaud (michel.billaud@laposte.net)

1er février 2022

Table des matières

1 Licence

- Les notes sont publiées dans https://www.mbillaud.fr/notes/
- Sources dans https://github.com/MichelBillaud/notes-diverses

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/33193/download

- recherche de la valeur et l'indice du maximum d'une liste
- recherche d'une sous-chaîne.

3 Recherche de la valeur et l'indice du maximum d'une liste

Sujet : Écrire une fonction `maxi` qui prend en paramètre une liste `tab` de nombres entiers et qui renvoie un couple donnant le plus grand élément de cette liste ainsi que l'indice de la première apparition de ce maximum dans la liste.

Exemple

```
>>> maxi([1,5,6,9,1,2,3,7,9,8])
(9,3)
```

Problèmes :

- le sujet oublie de préciser ce que la fonction doit retourner quand la liste est vide.
- pourquoi appeler `tab` une liste ?

3.1 Résolution

- On se permet donc de supposer qu'on n'appellera jamais `maxi` sur une liste vide.
- noter qu'on veut l'indice de la *première* occurrence.

Le calcul se fait en posant un résultat provisoire (valeur du minimum et de son indice), et en l'améliorant en utilisant les éléments suivants)

3.2 Solution

- Puisqu'on a besoin de retourner un indice, il n'est pas déraisonnable que la boucle porte sur les indices : utilisation d'un `range()`
- on peut prendre comme résultat provisoire le premier élément et son indice (O).

"Du coup", le `range` peut démarrer à 1, puisque l'indice 0 a déjà été pris en compte

```
def maxi(tab):
    valeur_maxi = tab[0]
    indice_maxi = 0
    for i in range(1, len(tab)):
        if tab[i] > valeur_maxi:
            valeur_maxi = tab[i]
            indice_maxi = i
    return (valeur_maxi, indice_maxi)
```

Ca serait mieux en renommant `tab` en `liste`, mais pour un examen, il vaut mieux s'en tenir à l'énoncé.

3.3 La question de la liste vide

Si on tient traiter le cas de la liste vide, il faut expliciter le choix qui est fait, parce que la spécification d'origine ne permet pas de le supposer. Exemple

```python
# Cette fonction retourne un couple (etc)
# La fonction retourne None (résultat = absence de couple)
# si la liste est vide

def maxi(tab):
    if len(maxi) == 0:
        return None
    ...
```

3.4 Bonus : automatiser les tests

Quand on programme, on commet des erreurs, que l'on détecte en faisant passer des tests.

C'est parfaitement normal de commettre des erreurs, mais c'est agaçant. Et faire passer des tests pour vérifier, c'est fatigant. Le résultat, c'est qu'on a tendance à ne pas les faire sérieusement, parce que

- c'est répétitif
- on se dit "ouais bon, non pas la peine ça marche je suis sûr",
- et le risque d'avoir une mauvaise nouvelle qui contredit ce bel optimiste est important.

Une solution à ce problème de *psychologie du travail* est d'automatiser autant que possible le passage des tests.

Un exemple simple, dans ce cas :

```python
def verifier_maxi(tab, attendu):
    print ("# appel maxi(" + str(tab) + ")")
    print ("- résultat attendu = " + str(attendu))
    print ("- resultat obtenu  = " + str(maxi(tab)))

# tests
verifier_maxi([1,5,6,9,1,2,3,7,9,8], (9,3))
verifier_maxi([5, 1, 2, 3], (5, 0))   #cas du premier
verifier_maxi([5, 1, 2, 5], (5, 0))   #cas du premier
verifier_maxi([11, 22, 33], (33, 2))  #cas du dernier
```

Dans l'appel à `verifier_maxi`, on fournit

- les données
- le résultat attendu.

La fonction `verifier_maxi` affiche les deux, ainsi que le résultat obtenu, dans un format qui facilite les vérifications. Exécution :

```
# appel maxi([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
- résultat attendu = (9, 3)
- resultat obtenu  = (9, 3)
# appel maxi([5, 1, 2, 3])
- résultat attendu = (5, 0)
- resultat obtenu  = (5, 0)
# appel maxi([5, 1, 2, 5])
```

```
- résultat attendu = (5, 0)
- resultat obtenu  = (5, 0)
# appel maxi([11, 22, 33])
- résultat attendu = (33, 2)
- resultat obtenu  = (33, 2)
```

On peut aussi prévoir l'affichage d'un message `ERREUR` bien visible si les résultats obtenus et attendus diffèrent.

4 Recherche d'une sous-chaîne dans une chaîne

Sujet résumé : compléter le code suivant pour qu'il indique si une chaîne (`gene`) apparaît dans

```
def recherche(gene, seq_adn):
    n = len(seq_adn)
    g = len(gene)
    i = ...                                    # 1
    trouve = False
    while i < ... and trouve == ... :          # 2
        j = 0
        while j < g and gene[j] == seq_adn[i+j]:
            ...                                # 3
        if j == g:
            trouve = True
        ...                                    # 4
    return trouve
```

Notes

- on ne peut pas dire qu'on soit gâtés par les notations choisies. On peut comprendre `g` pour la longueur de `gene`, mais ça ne paraît pas cohérent avec `n` pour la longueur de `seq_adn`.
- on induit facilement que `trouve` devrait être un booléen (on lui affecte `True` et `False`), la comparaison "`trouve == ...`" sent assez mauvais…

4.1 Résolution

Le plus simple est de repartir d'une description du résultat attendu :

> on dit que la chaîne `a` est dans la chaîne `b` si il existe un indice `i` tel que `a[0]==b[i]`, `a[1]==b[i+1]`, etc.

en allant jusqu'à la fin de `a`, c'est à dire

> `a[j]==b[i + j]`, pour tout `j` de 0 à len(a)-1‘

Il faut aussi tenir compte du fait que l'indice `i+j` doit être valide dans `b`, c'est à dire inférieur où égal à `len(b)-1`

De ces inégalités, on conclut que `i` doit être entre 0 et `len(b)-len(a)` compris.

Exemple : si les chaînes `a` et `b` sont respectivement de tailles 3 et 8, la dernière position possible pour `a[0]` est `i` = 5, `a` occuperait alors les positions d'indice

5, 6 et 7 dans `b` :

4.2 Solution

Identifions cette analyse et le code à trous.

La boucle `while` **intérieure** :

- sert visiblement à comparer les chaînes. Par miracle, les indices ont le même nom.
- elle doit s'arrêter dans deux cas : soit une différence a été détectée, soit toutes les comparaisons ont été faites
- le corps de boucle (trou 3) consiste à passer au `j` suivant `j += 1 # 3`
- si `j` a atteint la longueur de `gene`, la chaine `seq_adn` contenait bien une copie de `gene`, à partir de l'indice `i`

La boucle `while` **extérieure** :

- sert à essayer différents endroits où pourraient se trouver une copie de `gene` ;
- elle est contrôlée par la variable `i` :
  - qu'on fait partir de O dans le trou 1
  - qui doit augmenter de 1 à chaque tour (trou 4)
  - et qui doit s'arrêter quand la valeur `n-g` est dépassée
- une autre raison d'arrêter la boucle sur `i`, c'est si on a trouvé.

En se conformant strictement à l'énoncé, on obtient

```
def recherche(gene, seq_adn):
    n = len(seq_adn)
    g = len(gene)
    i = 0                                          # 1
    trouve = False
    while i < n-g+1 and trouve == False :          # 2
        j = 0
        while j < g and gene[j] == seq_adn[i+j]:
            j += 1                                 # 3
        if j == g:
            trouve = True
        i += 1                                     # 4
    return trouve
```

4.3 Solution nettoyée

On obtient une solution plus propre en appliquant quelques recettes de nettoyage

- nommer correctement la fonction
- choisir des notations cohérentes : `a`, `b` pour les chaînes,`la` et `lb` pour leurs longueurs, `ia` et `ib` pour les indices.
- tester correctement les booléens (`not` au lieu de comparaison par `==`)
- affecter le résultat d'une condition plutot que faire un `if`

```
# retourne True si et seulement si la chaîne a
# apparait dans la chaîne b
```

```
def est_contenue_dans(a, b):
    la = len(a)
    lb = len(b)
    ib = 0                                  # 1
    trouve = False
    while ib <= lb-la and not(trouve)       # 2
        ia = 0
        while ia < la and a[ia] == b[ib+ia]:
            ia += 1                          # 3
        trouve = (ia == la)
        ib += 1                              # 4
    return trouve
```

4.4 Décomposer pour simplifier

Une des règles d'or de la programmation est de se simplifier la vie en découpant en fonctions qui font peu de choses, mais le font bien.

```
def est_contenue_dans(a, b):
    for ib in range(len(b)-len(a)+1):
        if est_contenu_a_la_position(a, b, ib):
           return True
    return False

def est_contenu_a_la_position(a, b, ib):
    if i+len(a) > len(b):          # précaution
        return False
    for ia in range(len(a)):
        if a[ia] != b[ib+ia]:
            return False
    return True
```