Correction d'exercices sans préparation

Aux deux concours vous avez un exercice sans préparation en 10 minutes.

Au concours G2E c'est au tableau en revanche à l'Agro c'est au choix : au tableau ou sur l'ordinateur.

Quelques conseils de méthode (on est à l'oral : tout se verbalise) :

1. ***Comprendre l'énoncé*** *: le reformuler avec ses mots et l'illustrer sur un ou plusieurs exemples.*
2. ***Réfléchir avant de coder*** *: dégager la méthode, puis annoncer la démarche envisagée avant d'écrire la moindre ligne.*
3. ***Coder proprement*** *: commenter sa démarche au fur et à mesure et choisir des noms de variables parlants — l'examinateur évalue davantage le raisonnement que le code final.*
4. ***Vérifier*** *: dérouler le programme sur un exemple pour s'assurer qu'il renvoie bien le résultat attendu.*
   Pour ce dernier point la situation est très différente si vous avez choisi de coder au tableau ou sur l'ordinateur.

Exercice 1

1. Écrire une fonction `est_mixte` prenant en argument une liste composée de 0 et/ou 1 qui renvoie `False` si la liste est composée uniquement de 0 ou uniquement de 1 et `True` sinon.
2. Écrire une fonction `plus_longue_suite` prenant en argument une liste composée de 0 et de 1 qui renvoie la longueur de la plus longue suite de 1 de la liste.
   Exemple : si $L = [0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0]$, la fonction renvoie 3.

Réponse :

1. *On commence par expliquer la démarche sur un exemple au tableau :*

$$L = [0, 0, ..., 0, 0] \qquad L = [1, 1, ..., 1, 1]$$

On compare les éléments de L à sa première valeur.

```
def est_mixte(L):
    for k in range(1, len(L)):
        if L[k] != L[0]:
            return True
    return False
```

Version 2 : *On peut aussi penser à la somme qui donne le nombre de 1.*

```
def est_mixte(L):
    return 0 < sum(L) < len(L)
```

2. *On commence par expliquer la démarche sur un exemple au tableau :*

$$L = [0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, ..., 0, 1, 1, 1, 1, 1]$$

On utilise deux variables un compteur c qui va permettre de compter les 1 pour chaque série et m qui conserve le maximum courant.

```
def plus_longue_suite(L):
    c = 0
    m = 0
    for x in L:
        if x == 1:
            c += 1
        elif x==0 and c > 0:
            m = max(m, c)
            c = 0
    m = max(m, c)    # lorsque la liste se termine par une suite de 1
    return m
```

Exercice 2

1. Ecrire une fonction permettant de tester si une liste `L` est composée uniquement d'entiers compris au sens large entre `0` et `n-1`, avec `n` la longueur de `L` (hypothèse $\mathscr{H}$).
2. Ecrire une fonction permettant de tester si une liste `L` vérifiant l'hypothèse $\mathscr{H}$ est une permutation de $[\![0, n-1]\!]$.

Réponse :

1. *On commence par expliquer la démarche sur un exemple au tableau :*

$$L = [10, 2, 5..., 1, 2] \qquad L = [0, -1, 1, 2, 6, 5]$$

On teste que tous les éléments `x` de `L` vérifient `0 <= x < len(L)` .

```
def tester_H(L):
    for x in L:
        if not (0 <= x < len(L)):
            return False
    return True
```

Version 2 : *On peut aussi vérifier que ce sont des entiers.*

```
def tester_H(L):
        for x in L:
            if type(x) != int or not (0 <= x < len(L)):
                return False
        return True
```

2. *On commence par expliquer la démarche sur un exemple au tableau :*

$$L = [10, 2, 5..., 1, 3] \qquad L = [0, 3, 1, ..., 1, ..., 6, 5]$$

On vérifie qu' aucun élément `L[k]` n'apparaît déjà parmi les précédents `L[:k]` .

```
        def tester_permutation(L):
            if not tester_H(L):
                return False
            for k in range(len(L)):
                if L[k] in L[:k]:
                    return False
            return True
```

Version 2 : *Pour ceux qui connaissent `set(L)`.*

```
        def tester_permutation(L):
            return tester_H(L) and len(set(L)) == len(L)
```

Question 3. Bonus.

On suppose désormais que L est une permutation de $[\![0, n-1]\!]$: $L[i]$ désigne l'image de l'entier i.

Écrire une fonction `inverse` prenant en argument une telle liste L et renvoyant la permutation réciproque.

Réponse :

3. *On commence par expliquer la démarche sur un exemple au tableau :*
pour $L = [2, 0, 3, 1]$, on a $0 \mapsto 2$, $1 \mapsto 0$, $2 \mapsto 3$, $3 \mapsto 1$;
la réciproque envoie chaque image sur son antécédent, d'où `inverse`$(L) = [1, 3, 0, 2]$.

```
    def inverse(L):
        n = len(L)
        M = [0] * n
        for i in range(n):
            M[L[i]] = i
        return M
```

Vérification : appliquer deux fois la réciproque doit redonner la liste initiale, soit `inverse(inverse(L)) == L`.

Exercice 3

On s'intéresse ici aux vignettes auto-collantes vendus par paquets qui permettent de remplir un album.

1. Ecrire une fonction `paquet()` sans argument renvoyant aux hasard 3 entiers distincts entre 0 inclus et 100 exclu.
2. On considère un album de vignettes `A` modélisé par une liste de booléens.
   Par exemple : `[False, ..., False]` représente un album vide
   et `[True, ..., True]` représente un album plein.
   Ecrire une fonction `coller(paquet, album)` qui pour chaque numéro du `paquet` remplit la position correspondante dans l'`album`.
   Exemple : si `paquet = [3 , 2, 6]` on met `True` aux positions 2, 3 et 6.

Réponse :

1. *On commence par expliquer sur un exemple au tableau qu'on a bien compris la question :*

$$[2, 88, 9] \qquad [0, 8, 99] \qquad [2, 8, 2] \qquad [8, 8, 8]$$

On fait un tirage sans remise .

```
def paquet():
    S = []
    V = [ k for k in range(100) ]
    for i in range(3):
        a = rd.choice(V)
        V.remove(a)
        S.append(a)
    return S
```

Version 2 : *Pour ceux qui connaissent `rd.sample`.*

```
def paquet():
    V = [ k for k in range(100) ]
    return rd.sample(V, 3)
```

2. *On commence par expliquer ce qu'on va faire sur un exemple :*

si `paquet = [2,5,6]` on met `True` dans `album[2]` , `album[5]` , `album[6]`

```
def coller(paquet, album):
    for x in paquet:
        album[x] = True
```

Remarque : Cette fonction modifie la liste `album` passé en argument mais ne renvoie rien.

Version 2 : *Pour ceux qui n'aime pas les fonctions sans return.*

```
def coller(paquet, album):
    copie_album = album[:]
    for x in paquet:
        copie_album[x] = True
    return copie_album
```

Question 3. Bonus.

Écrire un programme permettant d'estimer le nombre moyen de paquets à acheter pour remplir l'album.

```
N = 10000
c = 0
for k in range(N):
    album = [False] * 100
    while not all(album):
        c += 1
        coller(paquet(), album)            # avec la version 1  de coller()
print(c/N)
```

Exercice 4

On travaille ici sur des chaines caractères ne contenant que : les caractères de la chaine :

`Alphabet = " abcdefghijklmnopqrstuvwxyz"`

1. Ecrire une fonction `rang(caractere)` qui retourne la position de `caractere` dans `Alphabet`.
2. Ecrire une fonction qui vérifie que deux chaines de caractères contiennent les mêmes caractères.
3. Ecrire une fonction qui vérifie que deux chaines de caractères sont des anagrammes l'une de l'autre.

Réponse :

1. *On commence par expliquer la démarche sur un exemple au tableau :*

`rang('a')` $= 1$ `rang('c')` $= 3$ `rang('z')` $= 26$

On parcourt Alphabet jusqu'à tomber sur le caractère, et on renvoie sa position.

```
def rang(caractere):
    for i in range(len(Alphabet)):
        if Alphabet[i] == caractere:
            return i
```

2. *On commence par expliquer la démarche sur un exemple au tableau :*

`"abba"` et `"ab"` $\rightarrow$ mêmes caractères `"abc"` et `"abd"` $\rightarrow$ non

On vérifie que chaque caractère de la première chaîne apparaît dans la seconde, puis ***réciproquement****.*

```
def memes_caracteres(s1, s2):
    for c in s1:
        if c not in s2:
            return False
    for c in s2:
        if c not in s1:
            return False
    return True
```

Version 2 : *Pour ceux qui connaissent* `set`*.*

```
def memes_caracteres(s1, s2):
    return set(s1) == set(s2)
```

3. *On commence par expliquer la démarche sur un exemple au tableau :*

`"chien"` et `"niche"` $\rightarrow$ anagrammes `"abba"` et `"ab"` $\rightarrow$ non

On compte le nombre d'occurrences de chaque lettre dans chaque chaîne (grâce à `rang`*) et on compare les deux tableaux d'effectifs.*

```
def anagrammes(s1, s2):
    compte1 = [0] * len(Alphabet)
    compte2 = [0] * len(Alphabet)
    for c in s1:
        compte1[rang(c)] += 1
    for c in s2:
        compte2[rang(c)] += 1
    return compte1 == compte2
```

Version 2 : *Pour ceux qui connaissent* `sorted`*.*

```
def anagrammes(s1, s2):
    return sorted(s1) == sorted(s2)
```