

QUELQUES REMARQUES SUR L'ÉPREUVE ITC CCINP PC 2024

1 Sur les fonctions, les listes et les dictionnaires

Il était demandé à plusieurs reprises dans l'énoncé de veiller à ce qu'une fonction modifie (ou non) un de ces arguments. On revient sur ce point.

Comportement des listes

Le point de départ est le fait que certaines structures de données (par exemple les listes et les dictionnaires) sont modifiables en python ("mutable"). Ce n'est pas le cas des chaînes de caractères. Par exemple si on définit une chaîne `chaine` par l'instruction `chaine = 'abc'`, l'instruction `chaine[1] = 'e'` n'est pas valable. On peut avoir l'impression de modifier `chaine` en écrivant `chaine = chaine + 'd'`, mais en fait une nouvelle chaîne (portant le même nom) est créée.

Pour les listes par contre, si l'on crée une liste par l'instruction `l = [0,1]`, puis l'on exécute la commande `l.append(2)` ou bien `l[1] = 3`, on ne crée pas une nouvelle liste, mais on modifie la liste existante. Les dictionnaires ont la même propriété.

Une autre particularité des listes est que si l'on déclare une première liste `l1 = [0,1]` puis qu'on exécute la commande `l2 = l1`, il y a une seule liste stockée en mémoire : si on modifie l'une, on modifie l'autre.

Tester :

```
l1 = [0,1]
l2 = l1
l2[0] = 3
print(l1)
```

Pour remédier à cela, on peut faire des copies terme à terme : `l3 = l1[:]`, `l4 = [a for a in l1]` ou utiliser la méthode `copy` (ce qui était suggéré dans l'énoncé). Tester :

```
l1 = [0,1]
l3 = l1[:] ; l4 = [a for a in l1] ; l5 = l1.copy()
l3[0] = 3 ; print(l3,l1)
l4[0] = 3 ; print(l4,l1)
l5[0] = 3 ; print(l5,l1)
```

On retiendra le fait que l'on nne crée jamais une liste par la commande `l_new = l`.

Exercice 1 – Cela pose également problème lorsque l'on veut créer un tableau de taille $n \times n$ contenant des 0.

Tester :

```
ligne = 4*[0]
tableau = 4*[ligne] ; print(tableau)
tableau[0][0] = 1 ; print(tableau)
```

Écrire une suite d'instructions qui crée un tableau (de même taille) en évitant le problème identifié ci-dessus. Tester.

Fonctions et listes

Le comportement des listes rentrées en paramètre d'une fonction est lié à ce qui précède. Quand une liste est un argument d'une fonction, les modifications effectuées lors de l'exécution de la fonction subsistent en dehors de la fonction ; on dit que la liste est passée "en référence" à la fonction.

Prenons des exemples pour fixer les idées. Tester :

```
def ajoutUn(x):
    x = x+1
    return x

x = 0
print(ajoutUn(x))
print(x)
```

La valeur de `x` n'a pas été modifiée. C'est sa valeur qui a été transmise à la fonction.

Remarque : la variable `x` créée à l'intérieur de la fonction est locale. La formulation de cette fonction est maladroite (voire trompeuse).

Observons maintenant ce qui se passe avec les listes :

```
def echange(l):
    l[0], l[1] = l[1], l[0]
    return l

l1 = [0,1]
print(echange(l1))
print(l1)
```

La liste `l1` a été modifiée : c'est sa référence, et non sa valeur, qui a été passée à la fonction.

On peut même remarquer que l'instruction `return` est superflue (voire trompeuse) : l'objectif de cette fonction est de modifier son entrée. On aurait dû écrire :

```
def echange2(l):
    l[0], l[1] = l[1], l[0]
```

Cette dernière fonction retourne `None`. Une telle fonction est parfois qualifiée (en référence à d'autres langages où la distinction est plus nette) de procédure ; on lira également que cette fonction agit par effet de bord.

Cette caractéristique peut paraître bizarre d'un première abord, mais dans de nombreuses situations on ne souhaite pas créer de nouvelle liste (penser aux tris par exemple) mais faire évoluer la liste d'entrée. Si l'on veut créer une nouvelle liste, il faut le faire explicitement.

Exercice 2 Écrire une fonction `echange3(l)` qui renvoie, sans modifier son entrée `l`, une liste similaire à `l` dans laquelle les deux premiers termes ont été échangés. On utilisera la méthode `copy` associée aux listes.

On peut également dans une fonction combiner le fait de retourner une certaine valeur et de modifier un paramètre d'entrée (d'où le terme "effet de bord") :

Exercice 3 – Écrire une fonction `echangeMaxMin(l)` qui prend en paramètre une liste d'entiers `l`, qui renvoie la différence entre le maximum et le minimum des valeurs de `l`, et qui échange ces deux valeur au sein de la liste. On supposera que la liste est de longueur au moins 2 et que tous

ses éléments sont distincts.

Par exemple, si `l1 = [1, 0, 4, 2]`, l'exécution de la commande `echangeMaxMin(l1)` devra renvoyer 4 et modifier `l1` de sorte que celle-ci soit égale à `[1, 4, 0, 2]`.

Écrire les instructions permettant de tester tout cela.

Avec les dictionnaires

Exercice 4 – Deux joueurs jouent au jeu suivant : ils lancent chacun un dé équilibré et ajoutent le résultat à leur score.

Rappel : on peut obtenir un nombre aléatoire selon une loi uniforme sur $[[a, b]]$ par la commande `rd.randint(a,b)` (après l'import : `import random as rd`).

On modélise un état du jeu par un dictionnaire `dicoJeu` ayant deux clefs : `dicoJeu['nbTours']` contient le nombre de tours joués et `dicoJeu['score']` contient le score sous la forme d'une liste à deux éléments (le premier éléments est le score du joueur 1 et le second le score du joueur 2).

1. Écrire une fonction `initJeu()` qui renvoie un dictionnaire représentant l'état initial du jeu.
2. Écrire une fonction `tourJeu(dicoJeu)` qui fait évoluer dictionnaire `dicoJeu` selon un tour du jeu.
3. Écrire une fonction `tourJeuBis(dicoJeu)` qui renvoie un dictionnaire représentant l'état du jeu après un tour de jeu à partir de l'état représenté par `dicoJeu`. Cette fonction ne devra pas modifier son entrée.
4. Écrire une suite d'instruction qui simule une partie durant 10 tours et qui affiche le nom du gagnant.

2 Sur SQL

2.1 Avec la BDD du DS

On reprend la base de données du DS.

Écrire les requêtes SQL permettant d'obtenir les informations suivantes.

1. Déterminer les noms des joueurs ayant joué une partie le 10 janvier 2024 (date représentée par la chaîne de caractères '10-01-2024') et commencé celle-ci.
2. Déterminer les noms des joueurs ayant joué une partie le 10 janvier 2024 (date représentée par la chaîne de caractères '10-01-2024').
3. Déterminer pour chaque joueur (identifié par son identifiant) le nombre de parties qu'il commencées et gagnées.
4. Déterminer les joueurs (leurs identifiants) dont le nombre de parties commencées et gagnées est supérieur à 10.

2.2 Avec la BDD Commerce

On reprend la base de données ayant servi de support au cours.

Autour de la question sur le pourcentage

1. Déterminer le pourcentage de commandes payées.
2. Déterminer le nombre de commande effectuées le '12/05/2015'.
3. Déterminer le nombre de commande effectuées le '12/05/2015' et qui sont payées.
4. Déterminer le pourcentage des commandes effectuées le '12/05/2015' qui sont payées.

Classement

1. Déterminer les trois pizzas les plus chères (classées en ordre descendant)
2. Déterminer la troisième pizza la plus chère

Autour de la question nécessitant un GROUP BY

1. Déterminer pour chaque pizza le nombre de fois où elle a été commandée.
2. Déterminer les pizzas ayant été commandées plus de 10 fois, ordonnées par nombre de commandes décroissant.