Interrogation d'ITC n°2

Semaine du 01/12/2025

Durée : 30 minutes

Nom :		Prénom :	

Consignes

- Les codes doivent être présentés en mentionnant explicitement l'indentation, au moyen par exemple de barres verticales sur la gauche.
- Pour qu'un code soit compréhensible, il convient de choisir des noms de variables les plus explicites possibles.
- La performance du code proposé à chaque question entrera dans l'évaluation, de-même que l'utilisation de boucles appropriées.
- Vous pouvez bien sûr utiliser une feuille de brouillon en parallèle.

/20

Exercice 1 Dictionnaires

On suppose construit un dictionnaire du type ci-dessous :

```
Notes = {"Jean DUPONT": ["MPSI1", 8.3], "Marie BOSSARD": ["MPSI2", 13.0], ...}
```

contenant les notes des deux classes de MPSI d'un célèbre lycée bordelais. Les clefs sont donc des chaînes contenant prénom, suivi d'un espace, et nom de l'étudiant, les valeurs sont des listes contenant deux objets : le premier est une chaîne contenant la classe, le second est la note obtenue.

1. /1 Écrire une (ou plusieurs) instruction(s) permettant de calculer le nombre d'étudiants total (dans les deux classes) contenu dans ce dictionnaire.

2. /1 Écrire une instruction permettant de diviser la note de `Jean DUPONT` par deux.

3. /1 On suppose que `Harry POTTER` ne fait pas encore partie des élèves. Écrire une instruction permettant d'ajouter `Harry POTTER` en `MPSI1`, avec la note de `20.0`.

4. /1 Écrire une instruction permettant de supprimer `Jean DUPONT` (et sa note/classe) du dictionnaire.

5. /3 Un professeur peu scrupuleux décide d'ajouter 1 point à tous les MPSI2 et de retirer 1 point à tous les MPSI1.
Écrire une fonction `trucage_notes(D:dict)->None` qui prend en argument un dictionnaire `D` du même type que `Notes`, et qui modifie en conséquence le dictionnaire. *On notera donc que cette fonction modifiera le dictionnaire `D` passé en argument mais ne renverra rien. Elle agit donc par « effets de bord ». On ne souciera pas des notes éventuellement négatives et dépassant 20.*

6. /1 Écrire une ou plusieurs instructions permettant (à taper dans une console) permettant d'observer l'effet de cette fonction sur le dictionnaire `Notes`.

Exercice 2 **Fonction mystère**

On considère la fonction suivante :

```
def inconnue(a:int, b:int, c:int)->int:
    if b == 0:
        return c
    else:
        return inconnue(a, b-1, c+a)
```

1. /1 Que renvoie l'instruction `inconnue(4, 3, 0)` ?

2. 2.1) /1 Conjecturer, et expliquer, ce que renvoie l'instruction `inconnue(n, m, 0)`.

   2.2) /2 Écrire une fonction itérative `f(n:int, m:int)->int` et renvoyant le même résultat que `inconnue(n, m, 0)`. La fonction ne devra pas utiliser le symbole `*`, c'est-à-dire le symbole de multiplication Python.

Exercice 3 **Compter le nombre de caractères communs**

On souhaite écrire une fonction `compte_commun(ch1:str, ch2:str)->int` qui renvoie le nombre de caractères en commun **aux mêmes indices** dans `ch1` et `ch2`, et qui renvoie `-1` dans le cas contraire. Par exemple,

- `compte_commun("ABC", "AAB")` renverra 1, `compte_commun("CAB", "ABC")` renverra 0,
- alors que `compte_commun("ABC", "AB")` renverra -1.

1. /2 Écrire une version itérative de `compte_commun`.

2. /3 Écrire une version récursive de `compte_commun`. On complètera le code ci-dessous.

```
def compte_commun_rec(ch1:str, ch2:str)->int:
    n1, n2 = len(ch1), len(ch2)
    if n1 != n2:
        return -1
    elif n1 == 0:
        return ____________
    else:
        if ch1[0] == ch2[0]:
            return ____________________
        else:
            return ____________________
```

Exercice 4 **Recherche dichotomique du maximum d'une liste** Pour déterminer le plus grand élément d'une liste $L = [\ell_0, \ell_1, \ldots, \ell_{n-1}]$ de n éléments, on peut utiliser la méthode dichotomique et récursive suivante, pour toute liste ayant au moins deux éléments :

- déterminer le maximum de la première moitié de la liste,
- déterminer le maximum de la seconde moitié de la liste,
- comparer les deux *maxima* et renvoyer le plus grand des deux.

Le cas de la liste à un élément est un cas trivial qui sert de cas terminal pour la programmation récursive.

Écrire une fonction `max_dicho` qui renvoie le maximum de la liste `L` selon ce principe.
La fonction ne devra pas se servir de la fonction `max` de Python

3

SOLUTIONS DES EXERCICES

Solution 1

```
>>> Notes = {"Jean DUPONT": ["MPSI1", 8.3], "Marie BOSSARD": \
↪ ["MPSI2", 13.0]}
```

1.
```
>>> Nb = len(Notes) # autre possibilité : compteur + boucle
>>> Nb
2
```
2.
```
>>> Notes["Jean DUPONT"][1] /= 2
>>> Notes
{'Jean DUPONT': ['MPSI1', 4.15], 'Marie BOSSARD': ['MPSI2', \
↪ 13.0]}
```
3.
```
>>> Notes["Harry POTTER"] = ["MPSI1", 20.0]
>>> Notes
{'Jean DUPONT': ['MPSI1', 4.15], 'Marie BOSSARD': ['MPSI2', \
↪ 13.0], 'Harry POTTER': ['MPSI1', 20.0]}
```
4.
```
>>> del Notes["Jean DUPONT"]
>>> Notes
{'Marie BOSSARD': ['MPSI2', 13.0], 'Harry POTTER': ['MPSI1', \
↪ 20.0]}
```
5.
```
def trucage_notes(D:dict)->None:
    for etudiant in D:
        classe = D[etudiant][0]
        if classe == "MPSI1":
            D[etudiant][1] -= 1
        else:
            D[etudiant][1] += 1
```
6.
```
>>> Notes
{'Marie BOSSARD': ['MPSI2', 13.0], 'Harry POTTER': ['MPSI1', \
↪ 20.0]}
>>> trucage_notes(Notes)
>>> Notes
{'Marie BOSSARD': ['MPSI2', 14.0], 'Harry POTTER': ['MPSI1', \
↪ 19.0]}
```

Solution 2

1. inconnue(4, 3, 0) → inconnue(4, 2, 4)) → inconnue(4, 1, 8))→ inconnue(4, 0, 12)) = $\boxed{12}$.
2. **2.1)** inconnue(n, m, 0) → inconnue(n, m-1, n) → inconnue(n, m-2, 2n) → inconnue(n, m-3, 3n) → ⋯ → inconnue(n, 0, mn) = $\boxed{m \times n}$.
   La variable b sert alors ici de « compteur récursif ».

   2.2)
```
def f(n:int, m:int)->int:
    S = 0
    for _ in range(m):
        S += n
    return S
```
```
>>> f(4, 3)
12
```

Solution 3

1.
```
def compte_commun(ch1:int, ch2:int)->int:
    n1, n2 = len(ch1), len(ch2)
    if n1 != n2:
        return -1
    else:
        nb = 0
        for i in range(n1):
            if ch1[i] == ch2[i]:
                nb += 1
        return nb
```
```
>>> compte_commun("ABC", "AAB")
1
>>> compte_commun("CAB", "ABC")
0
>>> compte_commun("ABC", "AB")
-1
```
2.
```
def compte_commun_rec(ch1:int, ch2:int)->int:
    n1, n2 = len(ch1), len(ch2)
    if n1 != n2:
        return -1
    elif n1 == 0: # n2 aussi nul dans ce cas
            return 0
    else:
        if ch1[0] == ch2[0]:
```

```
            return 1+compte_commun_rec(ch1[1:], ch2[1:])
        else:
            return compte_commun_rec(ch1[1:], ch2[1:])
>>> compte_commun_rec("ABC", "AAB")
1
>>> compte_commun_rec("CAB", "ABC")
0
>>> compte_commun_rec("ABC", "AB")
-1
```

Solution 4

```
def max_dicho(L:list):
    if len(L) == 1:
        return L[0]
    else:
        m = len(L)//2
        max_g = max_dicho(L[:m])
        max_d = max_dicho(L[m:])
        if max_g > max_d:
            return max_g
        else:
            return max_d
>>> max_dicho([-1, -3, 4, 5, 2])
5
>>> max_dicho([-1, -3, 4, 5])
5
```