Mini DS2 - Correction

Exercice 1. Cours

Q1. On dispose d'un dictionnaire `langages` dont les clés sont des noms de langages de programmation (type `str`), et la valeur associée est l'année de création de ce langage (type `int`). Une partie de ce dictionnaire est `langages = {'Fortran': 1954, 'SQL': 1974, 'C': 1972, ...}`.
Donner la commande permettant de :

a) Connaître le nombre d'éléments dans ce dictionnaire : `len(langages)`

b) Obtenir l'année de création de Python et la stocker dans une variable `py` :
`py = langages['Python']`

c) Rajouter dans ce dictionnaire le langage Go créé en 2009 : `langages['Go'] = 2009`

Q2. Aurait-on pu regrouper les langages créés la même année en choisissant comme clés des listes de chaînes de caractères (par exemple une clé `['Ada', 'C++']` associée à la valeur `1983`) ? Justifier.

Cela n'est pas possible car les clés d'un dictionnaire ne peuvent pas être des listes. Elles doivent impérativement être des objets immuables : entiers, flottants, chaînes de caractères ou tuples d'éléments de ces types.

Q3. Compléter la fonction `minimum` ci-dessous qui prend en argument un dictionnaire dont les valeurs sont des flottants (`float`) et qui renvoie la valeur minimale du dictionnaire. Par exemple `minimum({'a': 5.1, 'b': 3.2, 'c': 4.7})` doit renvoyer `3.2`.
On pourra utiliser `float('inf')` *pour créer un nombre correspondant à* $+\infty$.

```
def minimum(dico: dict) -> float:
    mini = float('inf')
    for cle in dico:
        if dico[cle] < mini:
            mini = dico[cle]
    return mini
```

Exercice 2. Retour sur le DS1

Q4. Écrire la commande permettant d'importer le module `random` et celle permettant d'importer **exclusivement** la fonction `cos` du module `math`.

```
import random
from math import cos
```

Q5. Compléter la fonction `ligne` ci-dessous qui prend en argument un entier n et un flottant x, et qui renvoie une liste de taille n dont le j-ème élément vaut $\cos(x + j)$.
On commencera par une assertion vérifiant que n est strictement positif.

```python
def ligne(n: int, x: float) -> [float]:
    assert n > 0
    line = []
    for j in range(n):
        line.append(cos(x + j))
    return line
```

On peut aussi utiliser une compréhension de liste :

```python
def ligne(n: int, x: float) -> [float]:
    assert n > 0
    return [cos(x+j) for j in range(n)]
```

Q6. En déduire une fonction `matrice` qui prend en argument un entier n, et qui renvoie une liste de n listes de taille n telles que le j-ème élément de la i-ème liste vaut $\cos(i+j)$.

On va créer la matrice ligne par ligne en réutilisant la fonction précédente.

```python
def matrice(n):
    mat = []
    for i in range(n):
        line = ligne(n, i)
        mat.append(line)
    return mat
```

Exercice 3. Cuisinons !

Dans tout cet exercice on choisit de représenter des recettes à l'aide de dictionnaires :

- les clés sont des chaînes de caractères (`str`) représentant les noms des ingrédients,
- les valeurs sont des entiers (`int`) correspondant à la quantité (toujours en grammes) demandée de cet ingrédient.

Q7. Écrire la ou les commande(s) permettant de créer un dictionnaire `smoothie` correspondant à la recette composée de 150 grammes de banane, 30 grammes de lait et 10 grammes de miel.

On peut soit le créer en une fois :

```python
smoothie = {'banane': 150, 'lait': 30, 'miel': 10}
```

soit initialiser un dictionnaire vide puis y ajouter chaque entrée une par une :

```python
smoothie = dict()  # ou {}
smoothie['banane'] = 150
smoothie['lait'] = 30
smoothie['miel'] = 10
```

Q8. Compléter la fonction `masse` ci-dessous qui prend en argument un dictionnaire représentant une recette et qui renvoie un entier représentant la masse totale du résultat de la recette.
Par exemple, `masse(smoothie)` doit renvoyer `190` (car $150+30+10=190$).

On reconnaît un classique algorithme de somme sur un dictionnaire.

```
def masse(recette: dict) -> int:
    total = 0
    for ingredient in recette:
        total = total + recette[ingredient]
    return total
```

Étant donné un stock d'ingrédients, on souhaite maintenant savoir si l'on peut réaliser une recette donnée. On choisit de représenter ledit stock par un dictionnaire ayant les mêmes caractéristiques que ceux pour les recettes. Par exemple si `stock = {miel: 100, lait: 1000, banane: 200}`, on peut réaliser la recette du smoothie décrite en **Q7**.

Q9. Compléter le tableau ci-dessous pour former un jeu de tests complet traitant les divers cas pouvant se produire :

Description cas	`stock`	`recette`	**Sortie**
Tout ce qu'il faut	`{'a': 1000, 'b': 100}`	`{'a': 30, 'b': 50}`	`True`
Tout juste ce qu'il faut	`{'a': 30, 'b': 50}`	`{'a': 30, 'b': 50}`	`True`
Pas assez d'un ingrédient	`{'a': 1000, 'b': 10}`	`{'a': 30, 'b': 50}`	`False`
Un ingrédient absent	`{'b': 100}`	`{'a': 30, 'b': 50}`	`False`
Ingrédient en stock non nécessaire	`{'a': 9, 'b': 9, 'c': 9}`	`{'a': 5, 'b': 8}`	`True`

On peut bien sûr imaginer encore d'autre cas comme un stock vide, aucune clé en commun entre les deux dictionnaires, etc. mais les quatre ci-dessus sont incontournables.

Q10. Écrire une fonction `possible` qui prend en argument deux dictionnaires `stock` et `recette` et qui renvoie un booléen valant `True` si on peut faire la recette avec le stock donné, et `False` sinon.

On va vérifier ingrédient par ingrédient s'il est possible de faire la recette. Le jeu de tests précédent met en valeur le fait qu'il y a deux choses à tester : a-t-on cet ingrédient en stock ? Puis a-t-on suffisamment de cet ingrédient ?

```
1 def possible(stock: dict, recette: dict) -> bool:
2     for ingredient in recette:
3         if ingredient not in stock:
4             return False
5         if stock[ingredient] < recette[ingredient]:
6             return False
7     return True
```

Exercice 4. Premières requêtes SQL

On dispose d'une table `Voies` qui répertorie des voies d'escalade sur une falaise donnée. Elle est composée des attributs suivants (entre parenthèses on précise le type de chacun) :

- `nom` (TEXT), clé primaire ;
- `longueur` (INT), en mètres ;
- `cotation` (TEXT), donnant la difficulté de la voie (il s'agit d'un nombre suivi d'une lettre).

Q11. Écrire une requête permettant d'obtenir tous les enregistrements de la table `Voies`.

```
SELECT *
FROM Voies
```

Q12. Écrire une requête permettant d'obtenir le nom et la longueur de chaque voie de difficulté 6a.

```
SELECT nom, longueur
FROM Voies
WHERE cotation = '6a'
```

Q13. Écrire une requête permettant d'obtenir les noms des voies de longueur supérieure ou égale à 20 mètres et dont la difficulté est 5c. Les résultats seront classés par ordre décroissant de longueur.

```
SELECT nom
FROM Voies
WHERE longueur >= 20 AND cotation = '5c'
ORDER BY longueur DESC
```