TP d'informatique n°4 bis : exercices supplémentaires sur les dictionnaires

PCSI 2026 – 2027

Ces exercices complètent le TP n°4. Ils supposent connues les opérations élémentaires sur les dictionnaires : création, ajout, accès, test d'appartenance et parcours avec `items()`. Sauf indication contraire, on n'utilisera pas de bibliothèque externe.

I Exercices de consolidation

♡ ***Exercice n°1*** Écrire une fonction `construit_index(texte:list)->dict` qui prend en argument une liste de mots et renvoie un dictionnaire `index` tel que `index[mot]` soit la liste des positions (indices) où le mot apparaît dans `texte`. Écrire aussi une fonction `positions(index:dict,mot:str)->list` qui renvoie la liste des positions d'un mot, ou une liste vide si le mot est absent.
Par exemple, avec

```
texte = ["le", "chat", "le", "chien", "le"]
```

on doit obtenir

```
index = {"le": [0, 2, 4], "chat": [1], "chien": [3]}
```

On pensera à créer la liste associée à un mot lorsqu'on le rencontre pour la première fois.

Exercice n°2 Écrire une fonction `a_doublon(L:list)->bool` qui renvoie `True` si un élément de L apparaît au moins deux fois, et `False` sinon.

1. Écrire une première version avec deux boucles **for** imbriquées.
2. Écrire une deuxième version avec un dictionnaire.
3. Comparer les temps d'exécution des deux versions pour des listes de taille 1000, 5000 et 10000. On pourra utiliser `time.perf_counter()`.

Par exemple :

```
a_doublon([3, 1, 4, 1, 5])  # True
a_doublon([3, 1, 4, 5])     # False
```

Exercice n°3 On représente les notes d'une classe par un dictionnaire de dictionnaires. Par exemple :

```
notes = {
    "Alice": {"maths": 15, "physique": 12, "info": 18},
    "Bob":   {"maths": 8,  "physique": 17, "info": 11},
    "Chloe": {"maths": 12, "physique": 14, "info": 16},
}
```

Écrire les fonctions suivantes :

- `moyenne(notes:dict,eleve:str)->float` qui renvoie la moyenne d'un élève, ou `None` si l'élève est inconnu ;
- `meilleur_eleve(notes:dict)->str` qui renvoie le nom du meilleur élève ;
- `moyenne_matiere(notes:dict,matiere:str)->float` qui renvoie la moyenne de la classe dans une matière, en ne tenant compte que des élèves qui ont une note dans cette matière (et `None` si personne n'en a).

Pour aller plus loin : on dispose d'un dictionnaire de coefficients

```
coef = {"maths": 4, "physique": 3, "info": 2}
```

Écrire `moyenne_ponderee(notes:dict,eleve:str,coef:dict)->float` qui calcule la moyenne pondérée d'un élève.

Exercice n°4 On dispose d'un dictionnaire `morse` qui associe à chaque lettre de l'alphabet son code Morse. On suppose les mots en minuscules et sans accent. Par exemple, `morse["a"]` vaut `".-"` et `morse["b"]` vaut `"-..."`.

1. Écrire une fonction `code(mot:str)->str` qui encode un mot en Morse, les codes des lettres étant séparés par une espace.
2. Écrire une fonction `decode(message:str)->str` qui décode un message Morse, en utilisant le dictionnaire inverse. On pourra réutiliser la fonction `inverse` vue dans le TP.

Par exemple :

```
code("chat")                    # "-.-. .... .- -"
decode("-.-. .... .- -")        # "chat"
```

II Exercices d'approfondissement

♡ ***Exercice n°5*** On considère une liste d'entiers L et un entier `cible`. Écrire une fonction `deux_sommes(L:list,cible:int)` qui renvoie un couple d'indices (i,j), avec $i<j$, tel que $L[i]+L[j]=$ cible, ou `None` s'il n'existe pas une telle paire. On demande une seule boucle **for** et l'utilisation d'un dictionnaire.
Par exemple :

```
deux_sommes([2, 7, 11, 15], 9)   # (0, 1)
deux_sommes([3, 2, 4], 6)        # (1, 2)
deux_sommes([3, 3], 6)           # (0, 1)
deux_sommes([1, 2], 5)           # None
```

Pour chaque élément x, on regardera si `cible-x` a déjà été rencontré. Quelle est la complexité de la version naïve avec deux boucles imbriquées ? Et celle de la version avec dictionnaire ?

Exercice n°6 Deux mots sont anagrammes s'ils contiennent exactement les mêmes lettres, avec le même nombre d'occurrences. Écrire une fonction `groupes_anagrammes(mots:list)->list` qui regroupe les mots anagrammes. Le résultat sera une liste de listes ; l'ordre des groupes et l'ordre des mots à l'intérieur d'un groupe n'ont pas d'importance.
Par exemple, avec `["chien","niche","chine","pomme","moche"]`, on doit obtenir, à l'ordre près :

```
[["chien", "niche", "chine"], ["pomme"], ["moche"]]
```

On pourra utiliser comme clé du dictionnaire le mot obtenu en triant les lettres. Attention : une liste n'est pas hashable, mais un tuple l'est.

Exercice n°7 On représente un polynôme par un dictionnaire `{degré: coefficient}` en omettant les coefficients nuls. Par exemple, `P = {0: 1, 2: 3}` représente $1+3x^2$, et le polynôme nul est représenté par `{}`.
Écrire les fonctions suivantes :

- `addition(P:dict,Q:dict)->dict` ;
- `multiplication(P:dict,Q:dict)->dict` ;
- `evaluation(P:dict,x:float)->float` ;

— `derivee(P:dict)->dict`.

On supprimera les coefficients nuls du résultat.
Par exemple, avec $P = 1 + 3x^2$ et $Q = 2x - 3x^2$:

```
P = {0: 1, 2: 3}
Q = {1: 2, 2: -3}
addition(P, Q)          # {0: 1, 1: 2}
multiplication(P, Q)    # {1: 2, 2: -3, 3: 6, 4: -9}
evaluation(P, 2)        # 13
derivee(P)              # {1: 6}
```

Pour l'addition, on pensera à utiliser `get(degre, 0)`. Pour la multiplication, on utilisera deux boucles **for** imbriquées sur les items de P et de Q.

Exercice n°8 Une matrice creuse est une matrice qui contient beaucoup de zéros. On la représente par un dictionnaire `{(i, j): valeur}` qui ne contient que les coefficients non nuls. Par exemple,

```
A = {(0, 0): 1, (0, 2): 2, (2, 1): 3}
```

représente la matrice

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix}.$$

Écrire les fonctions suivantes :

— `addition(A:dict,B:dict)->dict`;
— `transposee(A:dict)->dict`;
— `produit_matrice_vecteur(A:dict,v:list)->dict`
qui renvoie un dictionnaire `{i: valeur}` ne contenant que les coefficients non nuls du produit.

On supprimera les coefficients nuls du résultat de l'addition.
Par exemple, avec

```
B = {(0, 0): 4, (1, 1): 5, (2, 1): -3}
```

on doit obtenir :

```
addition(A, B)                          # {(0, 0): 5, (0, 2): 2, (1, 1): 5}
transposee(A)                           # {(0, 0): 1, (2, 0): 2, (1, 2): 3}
produit_matrice_vecteur(A, [1, 2, 3])   # {0: 7, 2: 6}
```

Pour aller plus loin : écrire `multiplication(A:dict,B:dict)->dict` pour le produit de deux matrices creuses.

III Pour aller plus loin

Exercice n°9 Pour un entier $n \geqslant 1$, on définit la suite de Syracuse par $u_0 = n$ et, pour tout $k \geqslant 0$,

$$u_{k+1} = \begin{cases} u_k/2 & \text{si } u_k \text{ est pair,} \\ 3u_k + 1 & \text{si } u_k \text{ est impair.} \end{cases}$$

On s'arrête lorsque $u_k = 1$. Écrire une fonction `longueur_syracuse(n:int)->int` qui renvoie le nombre d'étapes nécessaires pour atteindre 1. Par exemple, `longueur_syracuse(6)` doit renvoyer 8.
Pour accélérer les calculs, on utilise un dictionnaire global

```
cache = {1: 0}
```

qui mémorise les longueurs déjà calculées pour certaines valeurs.
Écrire une fonction `longueur_syracuse_cache(n)` qui renvoie la longueur de la suite en utilisant ce cache. Comparer les temps de calcul des deux versions pour tous les entiers de 1 à 10000.
Lorsqu'on tombe sur une valeur déjà présente dans le cache, on peut remplir le cache pour toutes les valeurs rencontrées depuis le départ. On pourra stocker ces valeurs dans une liste, puis les traiter à l'envers.

Exercice n°10 Écrire une fonction `plus_longue_sans_repetition(s:str)->int` qui renvoie la longueur de la plus longue sous-chaîne de s ne contenant pas deux fois le même caractère. Écrire aussi une fonction `plus_longue_sous_chaine(s:str)->str` qui renvoie une telle sous-chaîne (la première de longueur maximale).
Par exemple :

```
plus_longue_sans_repetition("abcabcbb")   # 3
plus_longue_sans_repetition("bbbbb")      # 1
plus_longue_sans_repetition("pwwkew")     # 3
plus_longue_sous_chaine("abcabcbb")       # "abc"
```

On utilisera deux indices `gauche` et `droite` et un dictionnaire qui mémorise la dernière position de chaque caractère. Quand on rencontre un caractère déjà vu après `gauche`, on déplace `gauche` juste après sa dernière occurrence.