N. S. I. Term. - DS02 (mardi 19 octobre 2021) – v3

Dictionnaires - Piles – Files

TIERS TEMPS : sauf exercice 2 Q7-Q8, exercice 4 Q2, exercice 5 Q3, exercice 6 Q2, exercice 8.

Exercice 1.

On considère le dictionnaire suivant, représentant les données relatives à un élève :

```
eleve1 = {'Nom' : 'BERNARD', 'Prénom' : 'Claude', 'Genre' : 'M',
          'Date de naissance' : ''}
```

Q1. Écrire une instruction, utilisant la fonction `print`, pour afficher, en utilisant les données enregistrées dans le dictionnaire `eleve1`, le prénom et le nom de cet élève, sous la forme : `Claude BERNARD`.

Q2. Écrire une instruction, pour que la date de naissance de cet élève, non renseignée initialement, soit mise à la valeur 06/12/2007 (sous la forme d'une chaîne de caractères).

Q3. Écrire une instruction pour supprimer du dictionnaire toute information relative au genre de l'élève (clé et valeur).

Q4. Écrire une instruction pour que l'information sur la classe de l'élève soit ajoutée au dictionnaire. L'élève sera supposé être scolarisée dans la classe nommée T5.

Q5. À l'aide de quelle(s) instruction(s), grâce à un parcours par les couples (clé, valeur), peut-on afficher toutes les informations sur l'élève.

Exercice 2.

Q1. [Préliminaire] Par quelles instructions peut-on construire une liste `L0` de longueur 10 dont tous les éléments sont égaux à zéro ? Proposer trois méthodes, utilisant respectivement l'opération « `*` » pour les listes, la méthode `.append()`, une compréhension de liste.

Q2. Rappeler l'instruction pour créer un dictionnaire vide, nommé `d`.

Q3. Rappeler l'instruction pour ajouter à un dictionnaire vide, `d`, une clé `'a'` associée à la valeur 0.

On considère la liste `L = ['a', 'b', …, 'z']` constituée des 26 lettres de l'alphabet, en minuscule.

Q4. À l'aide d'un parcours de la liste `L`, et des instructions rappelées aux questions **Q2** et **Q3**, construire, le dictionnaire `docc = {'a' : 0, 'b' : 0, …, 'z' : 0}` en ajoutant les couples (clé, valeur) au fil du parcours de `L` (l'algorithme de construction de ce dictionnaire présente des analogies avec celui qui permet de construire, avec la méthode `.append()`, la liste demandée en question **Q1**).

On souhaite compter les nombres d'occurrences de chaque lettre de l'alphabet dans un texte, à l'aide d'un dictionnaire, initialement égal au dictionnaire construit en **Q4**.

Pour chacune des 26 lettres de l'alphabet, la valeur associée à la clé correspondant à la lettre représentera le nombre d'occurrences de la lettre dans le texte. Ainsi, par exemple, initialement, `docc['a']` vaudra 0, et à la fin de l'application de l'algorithme de comptage des occurrences, si `docc['a']` vaut 10, cela signifiera que la lettre « a » apparaît 10 fois dans le texte.

On suppose que tous les caractères dans le texte sont, soit un espace, soit une des 26 lettres de l'alphabet, en minuscule.

Q5. On considère la fonction `compte_occ` dont le code (incomplet) est le suivant :

```
1. def compte_occ(texte):
2.     docc = {'a' : 0, 'b' : 0, …, 'z' : 0} ##ici le dictionnaire construit en Q4
3.     for ... : ## parcours de la chaîne « texte »
4.         if ... :
5.             docc[...] = ...
6.     return ...
```

Compléter le code de la fonction `compte_occ` pour qu'elle renvoie un dictionnaire donnant le nombre d'occurrence de chacune des 26 lettres de l'alphabet dans le texte.
On suppose que le dictionnaire `docc = {'a' : 0, 'b' : 0, …, 'z' : 0}` est construit et comporte bien 26 clés égales à chacune des 26 lettres de l'alphabet, associées à la valeur 0 (ne pas compléter la première ligne)

- Exemple de résultat d'un appel à la fonction :

```
>>> compte_occ('vive les vacances')
{'a' : 2, 'b' : 0, 'c' : 2, 'd' : 0, 'e' : 3, ..., 'z' : 0}
```

Q6. Si le nombre d'apparitions (occurrences) de la lettre `'a'` dans la chaîne `texte` est 10. Par quel calcul peut-on calculer la fréquence d'apparition de la lettre `'a'` dans la chaîne `texte` ?

Q7. Entre les lignes 5 et 6 du code de la fonction, ajouter des instructions pour que les valeurs dans le dictionnaire renvoyé, ne soient plus les nombres d'apparitions de chaque lettre dans le texte, mais la fréquence d'apparition de chaque lettre dans le texte.
On appliquera l'algorithme suivant :

- faire une copie du dictionnaire `docc` que l'on nommera `dfreq` ;
- à l'aide d'un parcours par les valeurs, déterminer le nombre de lettres dans la chaîne `texte` ;
- parcourir le dictionnaire `dfreq` pour changer les nombres d'apparitions en fréquences.

La fonction la fonction `compte_occ` pourra être renommée `calcule_freq` et renverra le dictionnaire `dfreq`.

Exercice 3.

Dans cet exercice, on ne demande aucun code Python.
Dans ce qui suit, les valeurs stockées dans la pile P et la file F sont des caractères (lettres majuscules 'A', 'B', 'C', …).
L'instruction « C = … » que l'on trouve ci-dessous indique que l'on affecte une valeur à une variable nommée C.
On indiquera sans ambiguïté, pour chaque question, la base et le sommet de la pile, l'entrée et la sortie de la file.

Q1. Donner, par l'intermédiaire de schémas, les états successifs de la pile P, au fil de l'exécution de la séquence d'instructions suivante :

1. creerPileVide(P)
2. empiler(P, 'A')
3. depiler(P)
4. empiler(P,'B')
5. C = depiler(P)
6. empiler(P,'A')
7. empiler(P,C).

Q2. Donner, par l'intermédiaire de schémas, les états successifs de la file F, au fil de l'exécution de la séquence d'instructions suivante :

1. creerFileVide(F)
2. enfiler(F,'A')
3. enfiler(F,'B')
4. enfiler(F,'C')
5. defiler(F);
6. C = defiler(F)
7. enfiler(F,'A')
8. defiler(F)
9. enfiler(F,'B')
10. enfiler(F,C).

Exercice 4.

Q1. Compléter ou corriger les quatre primitives du modèle de pile, basé sur des listes Python, vues en classe :

```
def creer_pile_vide() :
    ...
```

```
def est_vide_pile() :
    p = []
```

```
def empiler(p, x):
    ... append ...
```

```
def depiler(p, x):
    p.pop
```

Q2. Redonner les quatre primitives du modèle de file, basé sur des listes Python, vues en classe.

Exercice 5.

Dans cet exercice, on supposera définies les quatre primitives du modèle de pile de l'exercice 4.
Toutes les opérations sur les piles devront être réalisées à l'aide de ces quatre fonctions.

Q1. Écrire une séquence d'instructions, utilisant une boucle, permettant de créer une pile `p0` contenant les valeurs 0, 1, 2, 3, 4, 6, dans cet ordre, la valeur 0 se trouvant à la base de la pile.
Q2. Compléter, corriger ou modifier le code suivant afin que `vider_pile(p1)` vide la pile `p1` dans une nouvelle pile `p2`, renvoyée par la fonction.

```
def vider_pile(p1):
    ...
    while est_vide(p1):
        ...
        empiler(p2, ...)
    return p2
```

Que renvoie l'appel `vider_pile(p0)` où `p0` est la pile définie en question **Q1**.
Q3. Définir une fonction `copie_pile(p)`, prenant une pile `p` en argument et renvoyant une pile `pc` qui est une copie de la pile `p` en entrée. La pile `p` devra avoir retrouvé son état initial à la fin de l'application de l'algorithme. On devra utiliser une troisième pile `paux`, en plus de la pile `p` en entrée et de la pile `pc` que la fonction renvoie.

Exercice 6.

Dans cet exercice, on supposera définies les quatre primitives du modèle de pile de l'exercice 4.
Toutes les opérations sur les piles devront être réalisées à l'aide de ces quatre fonctions.

On suppose définie une pile `p`, contenant des entiers positifs.
Q1. Écrire un algorithme, sous la forme d'une séquence d'instructions en Python, permettant de vider la pile `p`, dans une pile `paux1` et une pile `paux2`, de sorte que `paux1` soit remplie avec les nombres pairs de `p` et `paux2` avec les nombres impairs de `p`.
Si p est la pile :

sommet	6
	4
	3
	10
	5
base	8

donner le schéma des piles `paux1` et `paux2` à la fin de l'exécution de l'algorithme.
Q2. Compléter l'algorithme pour qu'en fin d'exécution, la pile p contienne de nouveau les mêmes éléments qu'initialement, mais avec les nombres pairs tous au-dessous des nombres impairs (dans l'ordre initial).

Exercice 7. [exercice 63 du livre NSI terminale]

Dans cet exercice, on supposera définies les quatre primitives du modèle de pile de l'exercice 4.
Toutes les opérations sur les piles devront être réalisées à l'aide de ces quatre fonctions.

L'écriture *polonaise inverse* des expressions arithmétiques, place l'opérateur après ses opérandes. Cette notation ne nécessite aucune parenthèse ni aucune règle de priorité. Ainsi, eest une façon d'écrire celles-ci sans utiliser de parenthèses. Ainsi l'expression polonaise inverse décrite par la chaîne de caractères `'1 2 3 * + 4 *'` désigne l'opération traditionnellement notée $(1 + 2 \times 3) \times 4$. La valeur d'une telle expression peut être calculée facilement en utilisant une pile pour stocker les résultats intermédiaires. Pour cela, on observe un à un les termes de l'expression, et on effectue les actions suivantes :

- si on voit un nombre, on le place sur la pile ;
- si on voit un opérateur binaire ($+$ ou $\times$), on récupère les deux nombres au sommet de la pile, on leur applique l'opérateur, et on replace le résultat sur la pile.

Si l'expression était bien écrite, il y a bien toujours deux nombres sur la pile lorsque l'on voit un opérateur, et à la fin du processus il reste exactement un nombre sur la pile, qui est le résultat.

On suppose dans cet exercice que l'expression arithmétique à calculer ne comporte que des additions et des multiplications et est codée sous la forme d'une liste composée d'entiers et de symboles `'+'` ou `'*'`. Par exemple l'expression donnée en exemple ci-dessus sera codée par la liste

`L = [1, 2, 3, '*', '+', 4, '*']`.

Q1. Appliquer, « à la main » - sans coder, l'algorithme décrit ci-dessus à l'exemple `'1 2 3 * + 4 *'` en donnant les états successifs de la pile, initialement vide, au fur et à mesure du parcours de la liste L. Obtient-on bien le résultat attendu ?

Q2. Écrire une fonction, `calculeRPN`, prenant en argument une liste représentant une expression arithmétique en notation polonaise inverse et renvoyant la valeur de cette expression, à l'aide de l'algorithme décrit.
Si la liste ne correspond pas à une expression arithmétique correcte, la fonction doit renvoyer `None`.

Exercice 8.

Dans cet exercice, on supposera définies les quatre primitives du modèle de file de l'exercice 4.
Toutes les opérations sur les files devront être réalisées à l'aide de ces quatre fonctions.

On considère un jeu de 32 cartes représentée par une file constituée des entiers 1, 2, 3… 32, 1 étant la première valeur dans la file et 32, la dernière :

Entrée : 32 → 31 → … → 3 → 2 → 1 : Sortie.

Q1. Écrire une fonction `creer_jeu`, qui renvoie une file représentant un jeu de 32 cartes. On construira cette file à l'aide d'une boucle `for`.

On souhaite modéliser un jeu de bataille, avec deux joueurs, Alice et Bob.
On suppose que chaque carte l'emporte sur une autre si son numéro est supérieur au numéro de l'autre.
Chaque joueur reçoit au début du jeu 16 cartes.
Les cartes reçues par les deux joueurs sont modélisées respectivement par deux files, A et B.
On distribue les cartes en vidant la file, notée J, représentant le jeu, en ajoutant alternativement chaque carte défilée dans la file A et la file B. On suppose que les cartes ont été mélangées.

Q2. Écrire une séquence d'instructions réalisant la distribution des cartes.

Le jeu se déroule de la façon suivante, jusqu'à ce que le paquet de l'un des joueurs soit vide, chaque paquet fonctionnant comme une file :

- chacun des deux joueurs sort une carte de son paquet (file A ou file B)
- le joueur dont la carte est la plus forte, ramasse les deux cartes et les ajoute au fond de son paquet (dans un ordre quelconque).

Q3. Écrire une séquence d'instructions modélisant le déroulement d'une partie, jusqu'à ce qu'un vainqueur soit déclaré (un joueur perd lorsque son paquet de cartes est vide).