

Spécialité NSI Terminale

Contrôle n° 4

Sujet A

Lundi 12 décembre 2022

I Dictionnaires

Questions de cours (4 points)

1. Écrire l'instruction qui met de créer une nouvelle variable `personnage` et donc la valeur sera un dictionnaire vide.

2. Écrire les deux instructions qui ajoutent à ce dictionnaire `personnage` la valeur `'Jean'` associée à la clé `'Prénom'` et la valeur `'Valjean'` associé à la clé `'Nom'`.

3. a. Recopier et compléter le code suivant qui doit permettre d'obtenir un tableau de toutes les clés du dictionnaire `monDico`

```
tabCles = list(monDico....)
```

b. Recopier et compléter le code suivant qui doit permettre d'obtenir un tableau de toutes les valeurs du dictionnaire `monDico`

```
tabValeurs = list(monDico....)
```

4. On exécute le code suivant

```
capitales = {'Danemark' : 'Copenhague', 'Norvège' : 'Oslo',  
'Suède' : 'Stockholm', 'Finlande' : 'Helsinki'}  
  
for k in capitales :  
    print(k, capitales[k])
```

Que s'affiche-t-il dans la console ?

Exercice 1 (8 points)

Cet exercice porte sur les structures de données (dictionnaires).

La cryptographie est un ensemble de techniques permettant de chiffrer un message.

Une technique de cryptographie consiste à mélanger les lettres d'un alphabet et à réécrire le message avec ces permutations. En Python, on peut créer un dictionnaire dans lequel les clés sont les lettres de l'alphabet et les valeurs sont celles de l'alphabet mélangé.

Par exemple, si l'alphabet contient les 4 lettres A, B, C et D, et si le dictionnaire de l'alphabet mélangé est

```
alpha = {"A": "B", "B": "D", "C": "A", "D": "C"},  
la chaîne de caractères "BAC" sera chiffrée "DBA".
```

Un tel dictionnaire sera appelé dictionnaire de chiffrement.

1. On souhaite chiffrer un message écrit avec l'alphabet A, B, C, D, E, F, G à l'aide du dictionnaire

```
alpha = {"A": "B", "B": "D", "C": "A", "D": "C", "E": "F", "F": "G",  
"G": "E"}
```

a. Quelle est la valeur associée à la clé "D" ? En Python, comment l'obtenir?

b. Chiffrer la chaîne de caractères "BAGAGE" avec le dictionnaire `alpha`.

2. On considère qu'un mot est une chaîne de caractères (un objet de type `str`) écrite uniquement avec les 26 lettres de l'alphabet en majuscule. Par exemple, "ARBRE" est un mot et "L' ARBRE !" n'est pas un mot à cause des caractères : "'", " "(espace) et "!". Écrire une fonction `chiffrer(mot, alpha)` qui prend en paramètres `mot` un mot et `alpha` un dictionnaire de chiffrement, et qui renvoie une chaîne de caractères chiffrée avec le dictionnaire de chiffrement `alpha`.

3. On souhaite déchiffrer un mot chiffré avec cette méthode.

a. Si un mot est chiffré avec le dictionnaire de chiffrement `alpha = {"A": "B", "B": "D", "C": "A", "D": "C", "E": "F", "F": "G", "G": "E"}`, donner un dictionnaire permettant de le déchiffrer.

b. Écrire une fonction en Python appelée `dico_dechiffrement(dico)` qui prend en paramètre `dico` un dictionnaire de chiffrement et qui renvoie un dictionnaire permettant le déchiffrement. On pourra s'inspirer du code incomplet ci-dessous ou proposer une autre solution :

```
def dico_dechiffrement(dico):
    nouveau = {}
    for lettre in dico :
        code = dico[...]
        nouveau[...] = ...
    return nouveau
```

c. Écrire une fonction `dechiffre(mot, dico)` qui reçoit un mot chiffré et un dictionnaire de chiffrement et renvoie le mot décodé. On utilisera les fonctions écrites dans les questions précédentes.

4. On souhaite à présent créer un dictionnaire de chiffrement. Écrire une fonction `dico_chiffrement(alphabet)` qui prend en paramètre `alphabet` un tableau de lettres et qui renvoie un dictionnaire de chiffrement dont les clés sont les lettres du tableau `alphabet` et les valeurs sont les lettres du tableau `alphabet` mélangées.

On pourra utiliser la fonction `shuffle` du module `random` qui mélange en place un tableau. Par exemple, on a :

```
>>> tab = ["A", "B", "C", "D"]
>>> shuffle(tab)
>>> tab
["B", "A", "D", "C"]
```

II Piles et files

Exercice 2 (8 points)

Les interfaces des structures de données abstraites `Pile` et `File` sont proposées ci-dessous. On utilisera uniquement les **fonctions** ci-dessous :

Structure de données abstraite : `Pile`

Opérations

- `creer_pile_vide()`
renvoie une pile vide.
- `est_vide(P)`
renvoie `True` si la pile `P` est vide, `False` sinon.

- `empiler(P, elem)`
ajoute `elem` à la pile `P` et ne renvoie rien.
- `depiler(P)`
renvoie l'élément au sommet de la pile `P` en le retirant de la pile.

Structure de données abstraite : File

Opérations

- `creer_file_vide()`
renvoie une file vide.
- `est_vide(F)`
renvoie `True` si la file `F` est vide, `False` sinon.
- `enfiler(F, elem)`
ajoute `elem` à la file `F` et ne renvoie rien.
- `defiler(P)`
renvoie l'élément au sommet de la file `F` en le retirant de la file.

1. a. On considère la file `F` suivante :

enfilement -> `"rouge" "vert" "jaune" "rouge" "jaune"` -> défilement

Quel sera le contenu de la pile `P` et de la file `F` après exécution du programme Python suivant ?

```
P = creer_pile_vide()
while not est_vide(F) :
    empiler(P, defiler(F))
```

b. Recopier et compléter le code de la fonction `taille_file(F)` qui prend en paramètre une file `F` et qui renvoie le nombre d'éléments qu'elle contient. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

```
def taille_file(F) :
    F2 = ...
    cpt = ...
    while ... :
        enfiler(..., defiler(...))
        cpt = ...
    while ... :
        enfiler(..., defiler(...))
    return ...
```

2. Recopier et compléter une fonction `former_pile(F)` qui prend en paramètre une file `F` et qui renvoie une pile `P` contenant les mêmes éléments que la file. Le premier élément sorti de la file devra se trouver au sommet de la pile, le deuxième élément sorti de la file devra se trouver juste en-dessous du sommet, etc.

Exemple : si `F = "rouge" "vert" "jaune" "rouge" "jaune"` alors l'appel va renvoyer la pile `P` ci-dessous :

```

    "jaune"
    "rouge"
P =  "jaune"
    "vert"
    "rouge"
```

```
def former_pile(F) :  
    P1 = créer_pile_vide()  
    P2 = créer_pile_vide()  
    while ... :  
        empiler(..., defiler(...))  
    while ... :  
        elt = depiler(...)  
        enfiler(...)  
        empiler(...)  
    return ...
```

3. Écrire une fonction `nb_elements(F, elt)` **qui renvoie le nombre de fois où** `elt` **est présent dans la file** `F`.

Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

4. Écrire une fonction `verifier_contenu(F, nb_rouge, nb_vert, nb_jaune)` **qui renvoie** `True` **si** `"rouge"` **apparaît au plus** `nb_rouge` **fois,** `"vert"` **apparaît au plus** `nb_vert` **fois et** `"jaune"` **apparaît au plus** `nb_jaune` **fois dans la file** `F`. Elle renvoie `False` sinon. On pourra utiliser les fonctions précédentes.