Collections
Corrigé

Objectifs

— Savoir utiliser les ensembles et les dictionnaires
— Savoir lire un fichier
— Savoir récupérer une exception
— Raffiner (toujours !)

1 Compréhension du cours

Exercice 1 : Comprendre les ensembles

Compléter le programme `comprendre_ensemble.py` ci-après pour écrire les instructions Python qui réalisent les actions en commentaire. Les `assert` vérifient le résultat des fonctions demandées.

Solution :

```python
def test_comprendre_ensembles():
    # initialiser une variable nombres avec l'ensemble qui contient 1, 2, 3 et 2.
    nombres = {1, 2, 3, 2}

    # vérifier la taille de l'ensemble (le nombre d'éléments qu'il contient)
    assert len(nombres) == 3

    # vérifier si l'élément 2 est présent dans l'ensemble
    assert (2 in nombres) is True  # les parenthèses sont importantes (priorité des opérateurs
    assert 2 in nombres            # formulation à préférer : 'is True' est un pléonasme

    # vérifier si l'élément 5 est présent dans l'ensemble
    assert (5 in nombres) is False  # à éviter et préférer la formulation qui suit
    assert 5 not in nombres         # formulation à préférer

    # ajouter 33 dans l'ensemble.  Quelle est la taille de l'ensemble ?
    nombres.add(33)
    assert len(nombres) == 4   # taille de nombres
    assert 33 in nombres       # 33 est dans l'ensemble

```

```python
    # ajouter 2 dans l'ensemble. Quelle est sa taille ?
    nombres.add(2)
    assert len(nombres) == 4

    # supprimer l'élément 2 de l'ensemble.
    nombres.remove(2)   # l'élément doit être dans l'ensemble sinon KeyError
    nombres.discard(2)  # l'élément peut ne pas être dans l'ensemble

    # vérifier si 2 est encore dans l'ensemble
    assert 2 not in nombres

    # vérifier la taille de l'ensemble
    assert len(nombres) == 3

    # supprimer l'élément 7 de l'ensemble.
    try:
        nombres.remove(7)   # On aura KeyError car 7 n'est pas dans l'ensemble
    except KeyError:
        print('Pas trouvé')
    nombres.discard(7)      # Ne fait rien car 7 n'est pas dans l'ensemble


    # Soient e1 et e2 les deux ensembles suivants :
    e1 = {1, 2, 3}
    e2 = {2, 3, 4, 5}

    # intersection de e1 et e2
    assert {2, 3} == e1 & e2
    assert {2, 3} == e1.intersection(e2)

    # union de e1 et e2
    assert {1, 2, 3, 4, 5} == e1 | e2
    assert {1, 2, 3, 4, 5} == e1.union(e2)

    # les éléments de e1 qui ne sont pas dans e2 ?
    assert {1} == e1 - e2
    assert {1} == e1.difference(e2)

    # les éléments qui sont dans e1 ou e2 mais pas dans l'intersection
    assert {1, 4, 5} == e1 ^ e2
    assert {1, 4, 5} == e1.symmetric_difference(e2)
    assert {1, 4, 5} == (e1 | e2) - (e1 & e2)   # ok mais moins efficace !

    # créer un ensemble vide (appelé e)
    e = set()     #  Attention {} n'est pas un ensemble mais un dictionnaire

    assert isinstance(e, set)     # e doit être du type set (ensemble)
    assert len(e) == 0            # l'ensemble e est vide
    assert e1 == {1, 2, 3}        # e1 non modifié
    assert e2 == {2, 3, 4, 5}     # e1 non modifié

if __name__ == '__main__':
    test_comprendre_ensembles()
```

Exercice 2 : Comprendre les dictionnaires

Compléter le programme `comprendre_dictionnaire.py` : écrire les instructions Python qui réalisent les actions en commentaire. Les `assert` vérifient le résultat des actions demandées.

Solution :

```python
def test_comprendre_dictionnaire():
    # Les noms de pays en fonction de leur code ISO 3166-1 alpha-2
    pays = { 'FR': 'France', 'DE' : 'Allemagne',
             'ES' : 'Espagne', 'GB' : 'Angleterre' }

    # Obtenir le pays 'Allemagne'
    assert 'Allemagne' == pays['DE']

    # Vérifier que le code 'IT' n'est pas défini dans pays.
    assert not 'IT' in pays

    # Que se passe-t-il si on veut récupérer le pays associé à 'IT' ?
    try:
        italie = pays['IT']
        assert 'Italie' == italie
    except KeyError:
        print('Erreur ! IT non défini.')

    # Obtenir le pays qui correspond à un code donné ('IT' ou 'FR' par exemple).
    # S'il n'y a pas de pays associé au code on obtiendra 'INCONNU'.
    for code, nom_attendu in (('FR', 'France'), ('IT', 'INCONNU'),
            ('DE', 'Allemagne'), ('DK', 'INCONNU')):
        assert nom_attendu == pays.get(code, 'INCONNU')

    # Ajouter le nom 'Italie' associé au code 'IT'.
    pays['IT'] = 'Italie'
    assert 'IT' in pays            # le code 'IT' est défini
    assert 'Italie' == pays['IT']  # le pays associé est 'Italie'

    # Changer le pays associé à 'GB' pour mettre 'Royaume-Uni'
    pays['GB'] = 'Royaume-Uni'
    assert 'Royaume-Uni' == pays['GB'] # le pays associé à 'GB' est "Royaume-Uni'

    # Obtenir tous les codes connus dans pays
    codes = pays.keys()
    print('Les codes :', codes)

    # Obtenir tous les noms de pays connus dans pays
    les_pays = pays.values()
    print('Les pays :', les_pays)

    # Obtenir tous les couples (code, nom) de pays
    couples = pays.items()
    print('Les couples :', couples)

    # Afficher le contenu de pays sous la forme : code -> pays
    for un_code, un_pays in pays.items():
        print(f'{un_code} -> {un_pays}')
```

```python

    # Remarque : on peut le faire aussi ainsi :
    for un_code in pays:
        un_pays = pays[un_code]
        print(f'{un_code} -> {un_pays}')
    # mais c'est moins efficace car l'obtention du pays associé sera plus
    # coûteuse.  Si on a besoin et de la clé et de la valeur, il est préférable
    # d'utiliser items().


if __name__ == '__main__':
    test_comprendre_dictionnaire()
    print("ok")
```

2 Fichiers

Exercice 3 : Indexeur

On veut écrire un programme qui engendre automatiquement l'index d'un texte pour un ensemble de mots. Considérons par exemple le fichier `texte1.txt` suivant.

```
Ceci est un exemple de texte
On peut  par exemple  y indexer
les mots
    - exemple
    - texte
    - mot
    - mots
Un exemple reste un exemple et
ne couvre pas tous les cas
possibles
Le mot exemple apparaît deux fois
au début du paragraphe précédent
mais il ne faut pas voir deux fois
la ligne en question
```

Si on veut créer un index pour ce texte qui contient comme entrées les mots « exemple », « texte », « mot » et « mots », on utilisera la commande suivante :

```
python indexer.py texte1.txt exemple texte mot mots
```

Le résultat sera alors le suivant :

```
exemple : 1 2 4 8 11
mot : 6 11
mots : 3 7
texte : 1 5
```

On utilisera le module `sys` et plus particulièrement `sys.argv` qui est la liste des arguments de la lignes de commande. Le premier élément (à la position 0) est le nom du programme (`indexer.py`), le deuxième (position 1) est `texte1.txt`, etc.

3.1. Comment représenter dans le programme les mots à indexer ?
Solution : Un ensemble.
3.2. Comment représenter dans le programme l'index en cours de construction ?
Solution : Un dictionnaire avec comme clé un mot à indexer et comme valeur l'ensemble des numéros des lignes où le mot a été trouvé.
3.3. Écrire le programme `indexer.py` et l'utiliser pour reproduire l'exemple fourni.
Solution :

```
'''Définir un indexeur.'''

def indexeur(nom_fichier, mots):
    '''
    Indexer tous les mots du fichier du nom donné.

    :param nom_fichier: le nom du fichier à indexer
    :type nom_fichier: str
    :param mots: les mots à indexer dans le fichier (conteneur de str)
    :param: un ensemble de chaînes
    :return: le dictionnaire des mots et des numéros de ligne associés.
    '''
    with open(nom_fichier, 'r') as f:
        # normaliser les mots à indexer
        mots_normes = set()
        for mot in mots:
            mots_normes.add(mot.strip().lower())

        # Initialiser l'index
        index = {}
        for mot in mots_normes:
            index[mot] = set()

        # Indexer le texte
        for numero, ligne in enumerate(f, 1):
            # remplacer les caractères indésirables par des espaces
            for ignore in '.,\'"':
                ligne = ligne.replace(ignore, ' ')

            # construire l'ensemble des mots de la ligne
            mots_ligne = set(ligne.lower().split())

            # Ajouter ce numéro de ligne aux mots de la ligne à indexer
            for mot in mots_ligne & mots_normes :
                index[mot].add(numero)

        return index


def afficher_index(index):
    '''
    Afficher l'index, un dictionnaire dont les clés sont des chaînes de
    caractères et les valeurs des ensembles d'entier (numéro de ligne).

```

```
45      :param index: l'index
46      '''
47      for mot, numeros in sorted(index.items()):
48          # Afficher une entrée de l'index
49          #   Afficher le mot
50          print(mot, end=' :')
51
52          #   Afficher les numéros de lignes
53          for numero in sorted(numeros):
54              print('', numero, end='')
55          print()
56
57  def main(arguments):
58      '''Utiliser les arguments de la ligne de commande.'''
59      if len(arguments) <= 2:
60          print('Pas assez de paramètres !')
61          print(f'Usage : python {arguments[0]} nom_fichier mot1 mot2 ...')
62      else:
63          index = indexeur(arguments[1], arguments[2:])
64          afficher_index(index)
65
66
67  if __name__ == '__main__':
68      import sys
69      main(sys.argv)
```

3.4. Essayer de trouver des erreurs ou maladresses dans le programme en prenant d'autres exemples de textes et de mots à indexer. Par exemple, que se passe-t-il si on fait :

```
python indexer.py texte1.txt mot exemple mot texte mot mot
```

Solution : Voici quelques cas qu'il faudrait prendre en compte :

1. Le mot peut apparaître en majuscules et minuscules (sur la ligne de commande et dans le texte).
2. Un mot qui est une sous-partie d'une autre.
3. Un mot qui apparaît plusieurs fois sur la même ligne.
4. Il faudrait pouvoir traiter les caractères de ponctuation.
   La bonne solution serait d'utiliser les expressions régulières (module re en Python) que nous n'avons pas vues. La solution retenue est de remplacer les caractères de ponctuation par des espaces.