Conteneurs et fichiers

Objectifs

— Savoir utiliser les ensembles et les dictionnaires

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.

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

    # vérifier la taille de l'ensemble (le nombre d'éléments qu'il contient)
    assert ...

    # vérifier si l'élément 2 est présent dans l'ensemble
    assert ...

    # vérifier si l'élément 5 est présent dans l'ensemble
    assert ...

    # ajouter 33 dans l'ensemble.  Quelle est la taille de l'ensemble ?
    ...
    assert ...          # taille de nombres
    assert ...          # 33 est dans l'ensemble

    # ajouter 2 dans l'ensemble. Quelle est sa taille ?
    ...
    assert ...

    # supprimer l'élément 2 de l'ensemble.
    ...

    # vérifier si 2 est encore dans l'ensemble
    assert ...

    # vérifier la taille de l'ensemble
    assert ...
```

```
31
32      # supprimer l'élément 7 de l'ensemble.
33      ...
34
35
36      # Soient e1 et e2 les deux ensembles suivants :
37      e1 = {1, 2, 3}
38      e2 = {2, 3, 4, 5}
39
40      # intersection de e1 et e2
41      assert {2, 3} == ...
42
43      # union de e1 et e2
44      assert {1, 2, 3, 4, 5} == ...
45
46      # les éléments de e1 qui ne sont pas dans e2 ?
47      assert {1} == ...
48
49      # les éléments qui sont dans e1 ou e2 mais pas dans l'intersection
50      assert {1, 4, 5} == ...
51
52      # créer un ensemble vide (appelé e)
53      ...
54
55      assert isinstance(e, set)     # e doit être du type set (ensemble)
56      assert len(e) == 0            # l'ensemble e est vide
57      assert e1 == {1, 2, 3}        # e1 non modifié
58      assert e2 == {2, 3, 4, 5}     # e1 non modifié
```

Exercice 2 : Comprendre les dictionnaires

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

```
1   def test_commprendre_dictionnaire():
2       # Les noms de pays en fonction de leur code ISO 3166-1 alpha-2
3       pays = { 'FR': 'France', 'DE' : 'Allemagne',
4                'ES' : 'Espagne', 'GB' : 'Angleterre' }
5
6       # Obtenir le pays 'Allemagne'
7       assert 'Allemagne' == ...
8
9       # Vérifier que le code 'IT' n'est pas défini dans pays.
10      assert ...
11
12      # Que se passe-t-il si on veut récupérer le pays associé à 'IT' ?
13
14      ...
```

```python

    # 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')):
        assert nom_attendu == ...     # retrouver le nom associé à code

    # Ajouter le nom 'Italie' associé au code 'IT'.
    ...
    assert ...  # le code 'IT' est défini
    assert ...  # le pays associé est 'Italie'

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

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

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

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

    # Afficher le contenu de pays sous la forme : code -> pays
    ...

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

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 ?

3.2. Comment représenter dans le programme l'index en cours de construction ?

3.3. Écrire le programme `indexer.py` et l'utiliser pour reproduire l'exemple fourni.

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
```