

Collections

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.

```
1  def test_comprendre_ensembles():
2      # initialiser une variable nombres avec l'ensemble qui contient 1, 2, 3 et 2.
3      nombres = ...
4
5      # vérifier la taille de l'ensemble (le nombre d'éléments qu'il contient)
6      assert ...
7
8      # vérifier si l'élément 2 est présent dans l'ensemble
9      assert ...
10
11     # vérifier si l'élément 5 est présent dans l'ensemble
12     assert ...
13
14     # ajouter 33 dans l'ensemble. Quelle est la taille de l'ensemble ?
15     ...
16     assert ...          # taille de nombres
17     assert ...          # 33 est dans l'ensemble
18
19     # ajouter 2 dans l'ensemble. Quelle est sa taille ?
20     ...
21     assert ...
22
23     # supprimer l'élément 2 de l'ensemble.
24     ...
25
26     # vérifier si 2 est encore dans l'ensemble
```

```
27     assert ...
28
29     # vérifier la taille de l'ensemble
30     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}       # e2 non modifié
59
60     if __name__ == '__main__':
61         test_comprendre_ensembles()
```

Exercice 2 : Comprendre les dictionnaires

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

```
1  def test_comprendre_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    ...
15
16
17    # Obtenir le pays qui correspond à un code donné ('IT' ou 'FR' par exemple).
18    # S'il n'y a pas de pays associé au code on obtiendra 'INCONNU'.
19    for code, nom_attendu in (('FR', 'France'), ('IT', 'INCONNU'),
20                             ('DE', 'Allemagne'), ('DK', 'INCONNU')):
21        assert nom_attendu == ... # retrouver le nom associé à code
22
23    # Ajouter le nom 'Italie' associé au code 'IT'.
24    ...
25    assert ... # le code 'IT' est défini
26    assert ... # le pays associé est 'Italie'
27
28    # Changer le pays associé à 'GB' pour mettre 'Royaume-Uni'
29    ...
30    assert 'Royaume-Uni' == ... # le pays associé à 'GB' est "Royaume-Uni"
31
32    # Obtenir tous les codes connus dans pays
33    codes = ...
34    print('Les codes :', codes)
35
36    # Obtenir tous les noms de pays connus dans pays
37    les_pays = ...
38    print('Les pays :', les_pays)
39
40    # Obtenir tous les couples (code, nom) de pays
41    couples = ...
42    print('Les couples :', couples)
43
44    # Afficher le contenu de pays sous la forme : code -> pays
45    ...
46
47    if __name__ == '__main__':
48        test_comprendre_dictionnaire()
49        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.

```
1 Ceci est un exemple de texte
2 On peut par exemple y indexer
3 les mots
4     - exemple
5     - texte
6     - mot
7     - mots
8 Un exemple reste un exemple et
9 ne couvre pas tous les cas
10 possibles
11 Le mot exemple apparaît deux fois
12 au début du paragraphe précédent
13 mais il ne faut pas voir deux fois
14 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
```