Fonctions sur les conteneurs

November 21, 2019

1 Manipulation de conteneurs

Les conteneurs principaux sont: - les listes (et les tuples qui sont des listes non modifiables) - les dictionnaires - les ensembles

1.1 Fonctions génériques sur les conteneurs (voir mémento)

[2]:
```
d = {"le" : 34567, "la" : 5678, "les": 4691234, "arbre" : 67}
l = ["joe", 3, 5.78, [1,2,3]]
s = set(['a','e','i','o','u'])
```

[3]:
```
# longueur
len(d)
```

[3]: 4

[4]:
```
len(l)
```

[4]: 4

[5]:
```
# max et min
print(max(d))
print(max(l[3]))
```

```
les
3
```

[11]:
```
# somme des éléments (il faut qu'ils soient du même type et "additionnables")
print(sum(l[3]))
print(sum(l))
```

```
6
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)
```

```
        <ipython-input-11-8713e8ebe403> in <module>
          1 # somme des éléments (il faut qu'ils soient du même type et␣
↪"additionnables")
          2 print(sum(l[3]))
    ----> 3 print(sum(l))


        TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

[12]:
```
print(sorted(d))
print(d)
```

```
['arbre', 'la', 'le', 'les']
{'le': 34567, 'la': 5678, 'les': 4691234, 'arbre': 67}
```

[13]:
```
# sorted() crée une *copie* triée. La liste initiale n'est pas modifiée
L1 = [46, 7, 34, 98, 13, 8]
L2 = sorted(L1)
print(L1)
print(L2)
```

```
[46, 7, 34, 98, 13, 8]
[7, 8, 13, 34, 46, 98]
```

[14]:
```
# sort() modifie la liste à laquelle on l'applique ("tri sur place")
L1 = [46, 7, 34, 98, 13, 8]
print(L1)
L1.sort()
print(L1)
```

```
[46, 7, 34, 98, 13, 8]
[7, 8, 13, 34, 46, 98]
```

[22]:
```
# appartenance (pour les dictionnaires c'est de la clé qu'il s'agit)
print("le" in d)
```

```
True
```

1.2 Fonctions spécifiques aux listes

Deux fonctions pour “allonger” une liste: soit on ajoute un élément, soit on ajoute une liste d’éléments

[24]:
```
l.append("maison")
print(l)
```

```
['joe', 3, 5.78, [1, 2, 3], 'mapison', 'maison']
```

[15]:
```
L1.extend(L2)
print(L1)
print(L2)
```

```
[7, 8, 13, 34, 46, 98, 7, 8, 13, 34, 46, 98]
[7, 8, 13, 34, 46, 98]
```

[29]:
```
# Suppression d'un élément: remove() enlève la première occurrence.
# Exemple de boucle pour retirer toutes les occurrences
while 7 in L1:
    L1.remove(7)
    print(L1)
```

```
[8, 13, 34, 46, 98, 7, 8, 13, 34, 46, 98, 7, 8, 13, 34, 46, 98]
[8, 13, 34, 46, 98, 8, 13, 34, 46, 98, 7, 8, 13, 34, 46, 98]
[8, 13, 34, 46, 98, 8, 13, 34, 46, 98, 8, 13, 34, 46, 98]
```

1.3 Fonctions spécifiques aux dictionnaires

[34]:
```
print(d["le"])
d["un"] = 666
print(d)
```

```
34567
{'le': 34567, 'la': 5678, 'les': 4691234, 'arbre': 67, 'un': 666}
```

[37]:
```
del d["un"]
print(d)
```

```
{'la': 5678, 'les': 4691234, 'arbre': 67}
```

[39]:
```
print(d.get("le",0))
print(d["le"])
```

```
0


      ␣
↪----------------------------------------------------------------------------

       KeyError                                  Traceback (most recent call␣
↪last)

       <ipython-input-39-f0530c816f6c> in <module>
         1 print(d.get("le",0))
    ----> 2 print(d["le"])
```

```
KeyError: 'le'
```

Illustration de l'emploi de la méthode get(): d.get("le",0) renvoie la valeur trouvée dans le dictionnaire pour la clé "le", mais si jamais cette clé n'existe pas dans le dictionnaire, elle renvoie la valeur par défaut qui est son 2e paramètre (ici 0). Cette methode est couramment utilisée pour faire du code plus simple quand on remplit un dictionnaire en créant au fur et à mesure les entrée. Comparer les deux versions ci-dessous.

[17]:
```
texte1 = "Le narrateur rencontre dans un bus un jeune homme au long cou, coiffé␣
 ↪d'un chapeau mou orné d'une tresse tenant lieu de ruban. Ce quidam échange␣
 ↪quelques mots assez vifs avec un autre voyageur, puis va s'asseoir à une␣
 ↪autre place. Un peu plus tard, le narrateur revoit le même jeune homme cour␣
 ↪de Rome devant la gare Saint-Lazare en train de discuter avec un ami qui lui␣
 ↪conseille d'ajuster (ou d'ajouter) un bouton de son pardessus."
listemots = texte1.split()
dico = {}
# version 1 :
for m in listemots:
    pm = m.lower()
    dico[pm] = dico.get(pm,0) + 1

# version 2 :
for m in listemots:
    pm = m.lower()
    if pm in dico:
        dico[pm] = dico[pm] +1
    else:
        dico[pm] = 1

print("le dictionnaire comprend %d entrées" % len(dico))
print(dico)
```

```
le dictionnaire comprend 62 entrées
{'le': 3, 'narrateur': 2, 'rencontre': 1, 'dans': 1, 'un': 6, 'bus': 1, 'jeune':
2, 'homme': 2, 'au': 1, 'long': 1, 'cou,': 1, 'coiffé': 1, "d'un": 1, 'chapeau':
1, 'mou': 1, 'orné': 1, "d'une": 1, 'tresse': 1, 'tenant': 1, 'lieu': 1, 'de':
4, 'ruban.': 1, 'ce': 1, 'quidam': 1, 'échange': 1, 'quelques': 1, 'mots': 1,
'assez': 1, 'vifs': 1, 'avec': 2, 'autre': 2, 'voyageur,': 1, 'puis': 1, 'va':
1, "s'asseoir": 1, 'à': 1, 'une': 1, 'place.': 1, 'peu': 1, 'plus': 1, 'tard,':
1, 'revoit': 1, 'même': 1, 'cour': 1, 'rome': 1, 'devant': 1, 'la': 1, 'gare':
1, 'saint-lazare': 1, 'en': 1, 'train': 1, 'discuter': 1, 'ami': 1, 'qui': 1,
'lui': 1, 'conseille': 1, "d'ajuster": 1, '(ou': 1, "d'ajouter)": 1, 'bouton':
1, 'son': 1, 'pardessus.': 1}
```