1ère NSI

Représentation des données, types construits :

Ensembles et dictionnaires

Année Scolaire : 2020-2021

Les ensembles

Définition

On construit un ensemble en écrivant ses éléments entre accolades et en les séparant par des virgules. Cela ressemble à un tableau mais les éléments d'un ensemble, contrairement au tableau, ne sont pas ordonnés. En particulier, chercher à accéder au i-ième élément provoque une erreur.

```
>>> s = {2, 3, 5, 7}
```

l'ordre des éléments importe peu, on aurait pu écrire

```
>>> s = {5, 2, 7, 3}
```

les méthodes d'un ensemble

len

Comme pour un tableau, la taille d'un ensemble, c'est à dire son nombre d'éléments, est obtenue avec la fonction prédéfinie len.

```
>>> len(s)
4
```

on peut tester si un élément appartient à un ensemble avec la construction in de python, dont le résultat est un booléen

```
>>> 5 in s
True
```

```
>>> 4 in s
False
```

add-remove

On peut ajouter un élément avec add(...) ou le supprimer avec remove(...)

```
>>> s.add(42)
>>> s.remove(5)
>>> s
{2, 3, 42, 7}
```

On peut donc déclarer un ensemble vide, puis le remplir

```
>>> e=set()
>>> e.add(6)
>>> e.add(4)
>>> e.add(9)
>>> e
{4, 6, 9}
```

Il est possible de construire un ensemble en utilisant une construction par compréhension

```
>>> {x * x for x in range(10)}
{0, 1, 4, 9, 16, 25, 36, 49 ,64 , 81}
```

il est aussi possible de construire un ensemble à partir d'un tableau

```
>>> t = [3, 1, 2, 4, 5, 2, 3]
>>> s = set(t)
>>> s
{1, 2, 3, 4, 5}
```

l'ensemble est un élément qui ne comprend pas de doublons, comme un ensemble mathématique. D'ailleurs on peut construire des ensembles à partir des opérateurs d'union, d'intersection, ou encore de différence (des opérations ensemblistes)

```
>>> s = {1, 3, 9}
>>> u = {1, 2, 4, 8}
```

```
>>> s | u    #union
{1, 2, 3, 4, 8, 9}
>>> s & u   #intersection
{1}
>>> s - u    #différence
{3, 9}
>>> s ^ u    #ou exclusif
{2, 3, 4, 8, 9}
```

On peut parcourir tous les éléments d'un ensemble avec une boucle for

```
for x in s:
    print(x)
```

Les dictionnaires

Définition

Un dictionnaire en Python est une sorte de liste, mais au lieu d'utiliser des index, on utilise des clés alphanumériques que l'on va entrer au fur et à mesure.
L'ensemble des couples clés-valeurs sont enregistrés sous la forme *clé: valeur*, l'ensemble des couples étant séparés par une virgule et placés entre deux accolades.

Les **dictionnaires** se révèlent très pratiques lorsque vous devez manipuler des structures complexes à décrire et que les listes présentent leurs limites. Les dictionnaires sont des collections non ordonnées d'objets, c'est-à-dire qu'il n'y a pas de notion d'ordre (*i.e.* pas d'indice). On accède aux **valeurs** d'un dictionnaire par des **clés**.

Un dictionnaire est un conteneur mutable, ce ne sont pas des séquences, on ne peut pas accéder à leur contenu en donnant un indice.

Création et accès

```
>>> ani1 = {}   #ou ani1 = dict()
>>> ani1["nom"] = "girafe"
>>> ani1["taille"] = 5.0
>>> ani1["poids"] = 1100.0
>>> ani1
{'nom': 'girafe', 'taille': 5.0, 'poids': 1100.0}
```

En premier, on définit un dictionnaire vide avec les accolades { } (tout comme on peut le faire pour les listes avec [ ]).

On peut aussi le définir avec le mot-clé dict( ) comme list( ) pour les tableaux ou set( ) pour les ensembles.

Ensuite, on remplit le dictionnaire avec différentes clés ("nom", "taille", "poids") auxquelles on affecte des valeurs ("girafe", 5.0, 1100.0). Vous pouvez mettre autant de clés que vous voulez dans un dictionnaire (tout comme vous pouvez ajouter autant d'éléments que vous voulez dans une liste).

Remarque : un dictionnaire est affiché sans ordre particulier.

On peut aussi initialiser toutes les clés et les valeurs d'un dictionnaire en une seule opération :

```
>>> ani2 = {"nom": "singe", "taille": 0.70 , "poids": 1.75}
```

Accès, modification et parcours

Pour ajouter une clé et une valeur supplémentaire

```
>>> ani2["age"] = 15
>>> ani2
{'nom': 'singe', 'taille': 0.70, 'poids': 1.75, 'age': 15}
```

On modifie de la même façon la valeur associée à une clé existante

```
>>> ani2["age"] = 25
>>> ani2
{'nom': 'singe', 'taille': 0.70, 'poids': 1.75, 'age': 25}
```

Pour récupérer la valeur associée à une clé donnée, il suffit d'utiliser la syntaxe suivante dictionnaire["cle"]. Par exemple :

```
>>> ani1["taille"]
5.0e
```

Remarque : toutes les clés de dictionnaire utilisées jusqu'à présent étaient des chaînes de caractères. Rien n'empêche d'utiliser d'autres types d'objets comme des entiers.

Pour parcourir un dictionnaire on peut utiliser une boucle

Itération sur les clés :

```
>>> ani2 = {'nom':'singe', 'poids':0.70, 'taille':1.75}
>>> for cle in ani2.keys():
...     print(cle)
...
```

```
poids
nom
taille
```

Itération sur les valeurs :

```
>>> ani2 = {'nom':'singe', 'poids':0.70, 'taille':1.75}
>>> for val in ani2.values():
...     print(val)
...
0.70
singe
1.75
```

Itération sur le couple clés/valeurs

```
>>> ani2 = {'nom':'singe', 'poids':0.70, 'taille':1.75}
>>> for cle, val in ani2.items():
...     print(cle, val)
...
poids 0.70
nom singe
taille 1.75
```

Les mentions dict_keys et dict_values peuvent s'utiliser directement, ce sont des objets un peu particuliers. Si besoin, nous pouvons les transformer en liste avec la fonction list() :

```
>>> ani2.values()
dict_values(['singe', 0.70, 1.75])
>>> list(ani2.values())
['singe', 0.70, 1.75]
```

Pour vérifier si une clé existe dans un dictionnaire, on peut utiliser le test d'appartenance avec l'instruction in qui renvoie un booléen :

```
>>> if "poids" in ani2:
...     print("La clé 'poids' existe pour ani2")
```

```
...
La clé 'poids' existe pour ani2
```

Structure imbriquée

En créant une liste de dictionnaires qui possèdent les mêmes clés, on obtient une structure qui ressemble à une base de données :

```
>>> ani1 = {'nom': 'girafe', 'poids': 1100, 'taille': 5.0}
>>> ani2 = {'nom': 'singe','poids': 70, 'taille': 1.75}
>>> animaux = [ani1, ani2]
>>> animaux
[{'nom': 'girafe', 'poids': 1100, 'taille': 5.0}, {'nom': 'singe',
'poids': 70, 'taille': 1.75}]
>>>
>>> for ani in animaux:
...     print(ani['nom'])
...
girafe
singe
```

Vous constatez ainsi que les dictionnaires permettent de gérer des structures complexes de manière plus explicite que les listes.

Application

Application : les mots les plus fréquents dans un texte

Quel est le mot de sept lettres qui apparaît le plus souvent dans le texte du tour du monde en quatre-vingt jours de Jules Verne ?

Nous allons répondre à cette question avec un programme très simple qui utilise un dictionnaire.

Pour commencer il faut écrire une fonction occurrences qui prend en argument un tableau t, en renvoie un dictionnaire donnant pour chaque valeur apparaissant dans t, le nombre de fois qu'elle y apparaît.

```
def occurrences(texte: list) -> dict:
      d = dict()
      for lettre in texte:
            if lettre in d:
                  d[lettre] += 1
            else:
                  d[lettre] = 1
      return d
```

On part d'un dictionnaire vide. On parcourt ensuite le tableau texte. Pour chaque élément de texte on regarde s'il est déjà présent dans le dictionnaire. Si oui on incrémente la valeur associée, sinon cela signifie que c'est la première fois que l'on rencontre cette valeur, on ajoute alors cette entrée dans le dictionnaire avec la valeur 1.

Comment construire un tableau avec tous les mots du livre ?

On ouvre un fichier contenant le texte de l'œuvre, on en lit le contenu sous la forme d'une chaîne unique avec read(), puis on la transforme en un tableau de mots avec la fonction split() qui coupe le texte dès qu'il trouve un espace.

```
with open("ltdme80j-p.txt") as fich:
    texte = fich.read().split()
```

La fonction split() découpe la chaîne selon les espaces et les retours chariot.

Il ne reste plus qu'à appeler la fonction occurrences(t) pour obtenir le dictionnaire, puis à chercher dans celui-ci le mot de 7 lettres associé à la plus grande valeur.

La fonction plus_frequent(d, k)

```
def plus_frequent(d: dict, k: int) -> tuple:
      f = -1
      r = ""
      for mot in d:
      if len(mot) == k and d[mot] > f:
            resultat = mot
            f = d[mot]
      return resultat, f
```

le programme final :

```
def occurences(t: list) -> dict:
      """fonction qui compte le nombre d'occurences de chaque élément
contenu dans un tableau t"""
      d = dict()
      for lettre in texte:
          if lettre in d:
              d[lettre] += 1
          else:
              d[lettre] = 1
      return d

def plus_frequent(d: dict, nb_lettres: int) -> tuple:
      """fonction qui parcourt le dictionnaire (d) pour rechercher le
mot le plus fréquent de k lettres"""
      f = -1
      r = ""
      for mot in d:
          if len(mot) == nb_lettres and d[mot] > f:
              r = mot
              f = d[mot]
      return r, f

#ouverture du fichier contenant l'oeuvre
with open("ltdme80j-p.txt") as fich:
    texte = fich.read().split()

d = occurences(texte)

#affichage de chaque mot de i lettres le plus fréquent
for i in range(1,20):
      mot, freq = plus_frequent(d, i)
      print("le mot de {} lettre(s) le plus fréquent est {}, {} fois"
                                                .format(i, mot, freq))
```