Introduction à la Programmation 1 PYTHON

51AE011F

Séance 10 de cours/TD

Université Paris-Diderot

Objectifs:

— Les tableaux associatifs

— Les générateurs

[COURS]

Dans ce cours nous présentons quelques sujets un peu avancés de la programmation en Python.

1 Les tableaux associatifs

Pourquoi des tableaux associatifs [COURS]

— Nous avons dans ce semestre beaucoup travaillé avec les listes. On peut voir une liste d'entiers l comme une fonction qui a le domaine $\{n \mid 0 \leq n < len(l)\}$, et le codomaine $\mathbb{N}$. La liste `[73,17,42]`, par exemple, peut être vue comme une fonction avec le domaine $\{0, 1, 2\}$, et qui associe 73 à 0, 17 à 1, et 42 à 2. Il s'agit même des fonctions *modifiables*, car on peut *modifier* par une affectation la valeur contenue dans une case d'une liste.

— Puis nous avons vu que les listes sont en fait plus générales : on n'a pas seulement des fonctions vers les entiers, mais des fonctions vers n'importe quel autre type de valeurs. Par exemple, nous avons vu des listes de chaînes de caractères, des listes de listes d'entiers, etc.

— Les listes présentent quand même une restriction : le *domaine*, c'est-à-dire les indices d'une liste, sont toujours des entiers. Or, on souhaite parfois réaliser des fonctions avec un domaine différent, par exemple des chaînes de caractères : pensez par exemple à un annuaire qui associe au nom d'une personne un numéro de téléphone.

— Les *tableaux associatifs*, ou *dictionnaires* en anglais, permettent de réaliser des fonctions partielles dont le domaine est un ensemble fini de chaînes de caractères.

— Nous disons qu'un tableau associatif associe des *valeurs* à des *clefs*. Autrement dit, les clefs d'un tableau associatif d sont les éléments du domaine de d, vue comme une fonction partielle.

Comment travailler avec les tableaux associatifs [COURS]

— On peut créer un tableau associatif en listant tous les clefs, avec leurs valeurs associées. Par exemple

```
1 nom = {"Lennart": "Hofstaedter", "Sheldon": "Cooper", "Howard": "
    Wolowitz"}
```

défini un tableau associatif qui associe au trois prénoms *Lennart*, *Sheldon*, et *Howard* des noms de famille.

— En particulier, le tableau vide (qui n'a aucune clef) est noté `{}`.

— On peut obtenir la valeur associée par un tableaux associatif d à une clef c en écrivant $d[c]$. Par exemple, l'expression `nom["Lennart"]` donne comme résultat `"Hofstaedter"`.

— Dans le cas des listes il était facile de savoir si un entier i est un indice légal d'une liste l : il suffisait de tester si $0 \leq i < len(l)$. Cela n'est pas possible pour un tableau associatif dont les clefs peuvent former un ensemble quelconque. Pour cette raison, Python propose une expression particulière pour tester si une chaîne s est une clef d'un tableaux associatif t : `s in t`. Par exemple,

```
if ("Sheldon" in nom):
  print("on a  Sheldon")
else:
  print("pas de Sheldon !")

```

affiche `"on a Sheldon"`.

— `print` sait aussi afficher un tableau associatif. Par contre, l'ordre dans lequel un tableau est affiché n'est pas forcement l'ordre dans lequel on l'avait écrit. La raison pour cela est que l'ordre est simplement une question de représentation qui ne doit pas importer, et que Python choisit internement une représentation efficace qui peut utiliser un ordre différent.

Exercice 1 (Chercher dans des tableaux associatifs, $\star$)

Étant donnés un premier tableau associatif qui associe des noms à des prénoms de caractères, et un deuxième tableau associatif qui associe des noms d'acteurs à des noms de caractères :

```
nom = {"Lennart": "Hofstaedter", "Sheldon": "Cooper", "Howard": "
    Wolowitz"}
actor={"Cooper": "Parsons", "Hofstaedter": "Galecki"}
```

code/exoBigBang.py

1. *Écrire une fonction* `actor_of_character` *qui prend un prénom p en paramètre, et qui envoie le nom de l'acteur qui joue ce caractère. Par exemple,* `actor_of_character("Lennart")` *doit envoyer le résultat* `"Galecki"`.
2. *Améliorer votre fonction pour qu'elle envoie la chaîne* `"name unknown"` *dans le cas où p n'est pas un prénom connu d'un caractère, et envoie* `"actor unknown"` *dans le cas où le caractère est connu mais l'acteur qui le joue ne l'est pas. Par exemple,* `actor_of_character("Penny")` *doit envoyer* `"name unknown"`*, et* `actor_of_character("Howard")` *doit envoyer* `"actor unknown"`.

□

Exercice 2 (Un traducteur, $\star$)

Dans cet exercice nous allons écrire un traducteur français vers anglais très primitif. Étant donné un dictionnaire français-anglais, comme celui-ci :

```
dict={"langage": "language", "un": "a", "est": "is", "formidable": "
    wonderful"}
```

code/exoTranslator.py

Une phrase va être représentée par une liste de mots. Écrire une fonction `translate` *qui prend une phrase française en paramètre, et qui envoie une nouvelle phrase en anglais obtenue par une traduction mot par mot. Quand un mot français n'est pas trouvé dans le dictionnaire, on garde simplement le mot français. Ainsi,* `(translate(["Python", "est", "formidable"])` *doit donner le résultat* `["Python", "is", "wonderful"]`.

□

Plus sur les tableaux associatifs [Cours]

— On peut modifier une valeur dans un tableau associatif de la même façon qu'on modifie le contenu d'une case dans une liste. Par exemple,

```
chef = {"Stark": "Ned", "Lannister": "Tywin"}
chef["Stark"] = "Robert"
print(chef)

```

affiche

```
{"Stark": "Robert", "Lannister": "Tywin"}

```

— Une différence essentielle entre les tableaux associatifs et les listes vues ici en cours est le fait que les tableaux associatifs sont *extensibles*. Ça veut dire qu'on peut aussi faire une affectation dans un tableau associatif pour une clef qui n'existe pas encore : la nouvelle clef est alors ajoutée au tableau. Par exemple,

```
chef = {"Stark": "Ned", "Lannister": "Tywin"}
chef["Frey"] = "Walder"
print(chef)

```

affiche

```
{"Stark": "Ned", "Lannister": "Tywin", "Frey": "Walder"}

```

— En vérité, les tableaux associatifs peuvent aussi avoir des clefs qui sont d'autres types que des chaînes de caractère, mais c'est rarement utilisé, et nous n'avons pas encore vu les autres types qui peuvent être utilisés comme type de clefs.

— Dans d'autres langages de programmation, les tableaux associatifs sont souvent appelés des *tables de hachage*.

2 Les générateurs

Différents types de générateurs [Cours]

— Nous avons vu la construction `range(start,end,step)` dans le cadre des boucles `for`. Il est temps de ce demander quelle est la nature de l'objet construit par cette expression. Il s'agit d'un *générateur* : c'est un objet abstrait auquel on peut demander successivement de générer une nouvelle valeur. Il n'est pas nécessaire de savoir comment faire ces demandes à un générateur ; nous allons utiliser les générateurs seulement dans le contexte des boucles `for`, et ces boucles savent interagir avec un générateur pour obtenir de lui toutes les valeurs au fur et à mesure.
Il y a en Python des générateurs autres que `range` qui sont utiles à connaître, et qu'on peut utiliser à la place de `range` dans une boucle `for`.

— On peut utiliser une liste comme un générateur : les valeurs générées sont les éléments de la liste. Par exemple

```
for n in [2,3,4] :
  print(n*n)

```

affiche

4
9
16

— On peut utiliser une chaîne de caractères comme un générateur : les valeurs sont les caractères qui constituent la chaînes. Par exemple

```
s=""
for c in "miroir":
  s=c+s
print(s)
```

code/exampleStrings.py

affiche la chaîne "riorim"

— On peut lire toutes les lignes d'un fichier à l'aide d'un générateur `open` qui prend le nom du fichier en argument. Par exemple,

```
for l in open("exampleFiles.py"):
  print(l,end="")
```

code/exampleFiles.py

affiche simplement le contenu du fichier `exo1.py`, Attention, chaque ligne envoyée par le générateur contient aussi le saut de ligne final (quand il y en a un dans le fichier), pour cette raison l'exemple affiche la ligne avec un `end=""`.

— On peut aussi utiliser un tableau associatif comme générateur : les valeurs envoyées sont les clefs utilisées dans le tableaux, mais dans un ordre aléatoire. Par exemple

```
for c in {"hello" : "bonjour", "good bye" : "au revoir"}:
      print(c)
```

code/exampleKeys.py

affiche les deux lignes "hello" et "good bye", dans un ordre aléatoire.

Exercice 3 (Utiliser les générateurs, ⋆)

1. *Écrire une fonction qui prend une chaîne de caractères en paramètre, et qui envoie une copie de la même chaîne mais sans des sauts de ligne (caractère `"\n"` en python).*
2. *Écrire une fonction* `longueurs` *qui prend le nom d'un fichier en paramètre, et qui lit le fichier ligne par ligne. La fonction envoie un tableau associatif qui à chaque ligne trouvée dans le fichier associe sa longueur. Attention aux sauts de lignes qui ne doivent pas compter pour la longueur. Par exemple, si le fichier* `montexte` *a le contenu suivant :*

```
Longtemps,
je me suis
couché
de bonne heure.
```

code/montexte

alors l'appel `longueurs("montexte")` *doit envoyer le tableau associatif suivant :*

```
{"je me suis": 10, "Longtemps,": 10, "de bonne heure.": 15, "couché
   ": 6}

```

□

Exercice 4 (Travailler avec des dictionnaires, ⋆⋆)

1. *Écrire une fonction* `reverse` *qui prend un dictionnaire en paramètre, et qui renvoie un dictionnaire dans le sens inverse. Par exemple, appliquée au dictionnaire français-anglais suivant :*

```
{"maison": "house", "voiture": "car", "rue": "road", "car": "coach"
    }

```

la fonction doit renvoyer le dictionnaire anglais-français suivant :

```
{"car": "voiture", "road": "rue", "coach": "car", "house": "maison"
    }

```

2. *Écrire une procédure* `fauxAmis` *qui prend un dictionnaire en paramètre, et qui affiche tous les faux amis qui existent dans ce dictionnaire. Un faux ami dans un dictionnaire d'une langue source vers une langue cible est un mot du langage cible qui existe aussi dans la langue source, mais avec un sens différent. Par exemple, appliquée au dictionnaire français-anglais de la question précédente, la procédure doit afficher :*

   `Faux ami : car`

3. *Écrire une fonction qui prend deux dictionnaires en paramètre, et qui retourne le dictionnaire obtenu en appliquant d'abord la traduction selon le premier dictionnaire, puis sur le mot obtenu par la traduction selon le deuxième dictionnaire quand elle est possible. Par exemple, appliquée au dictionnaire français-anglais de la première question, et le dictionnaire anglais-allemand suivant :*

```
dictED={"house": "Haus", "road": "Strasse", "tower": "Turm"}

```

la fonction doit envoyer le dictionnaire français-allemand suivant :

```
{"maison": "Haus", "rue": "Strasse"}

```

□