Licence de Mathématiques Année 2014-2015
Université d'Angers

DICTIONNAIRES ET CLASSES EN PYTHON

Les dictionnaires

Les dictionnaires constituent un type d'objet Python qui ressemble aux listes (ils sont modifiables comme elles), mais ce ne sont pas des séquences. Les éléments que nous allons y enregistrer ne seront pas disposés dans un ordre particulier. En revanche, nous pourrons accéder à n'importe lequel d'entre eux à l'aide d'un index spécifique que l'on appellera une **clé**, souvent alphabétique ou numérique. Comme dans une liste, les éléments mémorisés dans un dictionnaire peuvent être de n'importe quel type (valeurs numériques, chaînes de caractères, listes,... et même dictionnaires).

Nous allons créer un premier dictionnaire dont les clés sont des chaînes. Puisque le type dictionnaire est modifiable, nous allons le remplir au fur et à mesure. On reconnaît le type dictionnaire aux accolades qui délimitent ses éléments:

```
jack={}
jack["sexe"]="M"
jack["age"]=22
jack["redoublant"]=False
jack["notes"]=[17,11,7,2]
print jack
print jack.keys(), jack.values()
print jack.has_key("age")
print jack.has_key("Amis")
print jack.items()
peter=jack
paul=jack.copy()
del jack["notes"]
print jack, peter, paul
```

Cela dit, on peut aussi construire un dictionnaire dont les clés sont des listes, par exemple un tuples. Cela permet par exemple de placer des arbres dans une forêt.

```
foret={}
foret[(1,1)]="peuplier"
foret[(1,3)]="chene"
foret[(3,1)]="hetre"
foret.get((1,1),"rien")
foret.get((2,2),"rien")
```

Parcours d'un dictionnaire Pour parcourir un dictionnaire, on peut utiliser une simple boucle for, qui affectera à la variable de travail les clés du dictionnaire dans un ordre imprévisible. La synthaxe est la suivante :

```
panier={"pommes":7,"salade":1,"couleur:"Jaune"}
for clef in panier:
    print clef, panier[clef]
```

On peut aussi utiliser la méthode items() utilisées précedemment qui construit une liste de tuples de la manière suivante :

```
panier={"pommes":7,"salade":1,"couleur:"Jaune"}
for clef,valeur in panier.items():
    print clef, valeur
```

Exercice d'application Les dictionnaires permettent de construire des histogrammes de façons très élégantes. On va étudier la fréquence d'utilisation de chacune des lettres de l'alphabet dans un texte donné par l'utilisateur.

1. Grâce à la commande raw_input(), demander à l'utilisateur d'écrire une phrase sans accent et affecter cette phrase à la variable texte.

2. Créer un dictionnaire frequence_lettres dont les clés sont les lettres de l'alphabet et les valeurs le nombre d'occurences de chaque lettre dans la variable texte.
   N.B.: utiliser une boucle sur la chaîne de caractères texte.

3. Afficher le résultat par ordre décroissant d'apparition de chaque lettre, puis par ordre alphabétique, grâce à la fonction sorted() (attention, cette méthode ne s'applique qu'aux listes).

4. On veut aller un peu plus loin et construire à l'aide de Python un générateur de "word cloud" ou nuage de mots. Pour cela, on va procéder comme ci-dessus mais cette fois-ci en comptant la fréquence d'utilisation des mots dans un fichier texte.

   (a) Ecrire un texte de quelques phrases et remplacer tous les caractères non-alphabétiques par des espaces grâce à une boucle.

   (b) Convertir le contenu du texte en liste de mots à l'aide de la méthode split(). Attention, cette fonction ne s'applique qu'aux chaines de caractères.

   (c) Compter le nombre d'occurences de chaque mot et garder les mots dont l'occurence est supérieure à 5, et dont la longueur est supérieure à 3 caractères.

Les classes

Les classes sont une partie cruciale de Python, langage de programmation orienté objet. Une classe est un objet Python qui possède :

- ses propres variables, appelées **variables membres**, communes à tous les membres de la classe.

- ses propres fonctions, appelées **méthodes**, qui agissent sur tous les éléments de la classe.

La méthode __init__() est la méthode qui permet d'initialiser vos objets. Le premier argument est toujours self qui fait référence à l'objet, et les arguments supplémentaires permettent d'ajouter des variables membres qui seront initialisées à la création de chaque objet. Dans l'exemple ci-dessous, nous créons une classe Animal ayant 4 variables membres et deux méthodes:

```
class Animal(object):
    est_vivant=True
    sante="Bonne"
    def __init__(self,nom,age):
        self.nom=nom
        self.age=age
    def description(self):
        print self.nom, self.age
chat=Animal("Khemize",7)
chien=Animal("Bambou",2)
print chat.nom, chat.est_vivant
chien.description()
```

On va tout d'abord ajouter quelques méthodes et variables à notre classe membre :

1. Ajouter une méthode chasseur() qui tue l'objet de la classe Animal en modifiant sa variable est_vivant.

2. Ajouter une variable sauvage à votre classe qui renseigne si l'animal est sauvage (ou domestique). Modifier la méthode chasseur() de manière à sauver les animaux domestiques.

3. Modifier la méthode description() de manière à afficher si l'animal est sauvage ou domestique, et s'il est sauvage, son état (vivant ou tué).

4. Ajouter une variable membre predateur ayant pour valeur par défaut une liste vide []. Construire une méthode danger() qui permet d'ajouter un prédateur à la variable predateur.

La notion d'héritage Les classes peuvent être ordonnées entre elles grâce à la notion d'héritage. par exemple, une classe peugeot qui hérite d'une classe voiture prend les attributs et méthodes de la classe voiture. Le code ci-dessous construit une classe insecte qui hérite de la classe Animal.

```
class Insecte(Animal):
    sauvage=True
    def __init__(self,nom,je_pique):
        self.nom=nom
        self.je_pique=je_pique
    def protege(self):
        self.je_pique=False
moustique=Insecte("bzz",True)
print moustique.nom, moustique.est_vivant
moustique.protege()
print moustique.je_pique
```

Exercice d'application On peut maintenant créer un programme Python qui va simuler l'évolution d'une partie de chasse dans la forêt où les animaux sauvages affrontent les chasseurs.

1. Créer une liste d'une petite douzaine d'animaux sauvages (biches, sangliers, bécasses, lièvre), quelques chiens de chasses et quelques insectes (moustiques, abeilles, mouches).

2. On considère que la probabilité p qu'un chasseur tue un animal dépend du type d'animal (sauvage ou domestique) et de son taux d'alcoolémie (x en g/L) par la formule $p = (1+x)^{-1}$ si l'animal est sauvage, et $p = \mathbb{1}_{x\geq 1}(x-1)/x$ si l'animal est domestique. Modifier la méthode chasseur() définie précédemment.

3. Créer une fonction etat_des_lieux() qui recence à un instant donné, les animaux présents dans la forêt et leur état (mort ou vivant).

4. Simuler la première partie de chasse où chaque animal se présente une fois devant un chasseur dans un ordre aléatoire. Un chasseur tire, et tous boivent un coup (+0.15g/L) après chaque passage d'un animal. Afficher l'état des lieux à l'issue de cette première partie de chasse.

5. Maintenant les insectes rescapés vont piquer les chasseurs, selon la valeur de leur variable membre je_pique. Chaque piqûre divise par deux la probabilité que le chasseur touche sa cible. Ajouter une variable à la méthode chasseur en conséquence. Simuler la deuxième partie de chasse avec les animaux sauvages rescapés. A chaque passage d'un animal, les chasseurs boivent un coup et se font piquer par un insecte avec proba 1/2. Afficher l'état des lieux à l'issue de cette deuxième partie de chasse.