Les Dictionnaires Python

Chapitre 9

Python for Informatics: Exploring Information

www.pythonlearn.com

Qu’est-ce qu’une Collection?

- Une collection est pratique car elle nous permet de transporter plusieurs valeurs dans un même paquet.
- Nous avons un paquet de valeurs dans une seule “variable”
- Nous faisons cela en ayant plus d’une place dans la variable
- Nous avons des moyens de trouver les différentes places dans la variable

Qu'est-ce qui n'est pas une "Collection"

La plupart de nos variables contiennent une valeur - lorsque nous ajoutons une autre valeur à notre variable, l'ancienne valeur est remplacée

```
$ python
Python 2.5.2 (r252:60911, Feb 22 2008, 07:57:53)
[GCC 4.0.1 (Apple Computer, Inc. build 5363)] on
darwin
>>> x = 2
>>> x = 4
>>> print x
4
```

L’histoire des deux Collections…

- La Liste
  - › Une collection linéaire de valeurs qui restent ordonnées

- Le Dictionnaire
  - › Un “paquet” de valeurs ayant chacune leur propre étiquette

Dictionnaires

http://en.wikipedia.org/wiki/Associative_array

Les Dictionnaires

- Les Dictionnaires sont les collections de données les plus puissantes de Python
- Les Dictionnaires nous permettent de faire rapidement des opérations de type base de données dans Python
- Les Dictionnaires ont des noms différents selon les différents langages
  - Associative Arrays - Perl / PHP
  - Properties ou Map ou HashMap - Java
  - Property Bag - C# / .Net

http://en.wikipedia.org/wiki/Associative_array

Les Dictionnaires

- Les Listes indexent leur entrée selon leur position dans la liste
- Les Dictionnaires sont comme des sacs - désordonnés
- Donc nous indexons ces diverses choses que nous mettons dans un dictionnaire grâce au “lookup tag”

```
>>> sac = dict()
>>> sac['argent'] = 12
>>> sac ['bonbons'] = 3
>>> sac ['mouchoirs'] = 75
>>> print sac
{'argent': 12, 'mouchoirs': 75, 'bonbons': 3}
>>> print sac ['bonbons']
3
>>> sac ['bonbons'] = sac ['bonbons'] + 2
>>> print sac
{'argent': 12, 'mouchoirs': 75, 'bonbons': 5}
```

Comparons les Listes aux Dictionnaires

- Les Dictionnaires sont comme les Listes sauf qu'ils utilisent des clés au lieu des nombres pour rechercher des valeurs

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst[21, 183]
>>> lst[0] = 23
>>> print lst[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['cours'] = 182
>>> print ddd
{'cours': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'cours': 182, 'age': 23}
```

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst
[21, 183]
>>> lst[0] = 23
>>> print lst
[23, 183]
```

Liste

Clé	Valeur
[0]	21
[1]	183

lll

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['cours'] = 182
>>> print ddd
{'cours': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'cours': 182, 'age': 23}
```

Dictionnaire

Clé	Valeur
['cours']	183
['age']	21

ddd

Les Littérales de Dictionnaires (Constants)

- Les Littérales de Dictionnaires sont délimitées par des accolades et comportent une listes de clés : des paires de valeurs
- Vous pouvez créer un dictionnaire vide à l'aide d'accolades vides

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print jjj
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print ooo
{}
>>>
```

Quel est le nom le plus fréquent?

marquard

zhen

csev

zhen

cwen

marquard

zhen

csev

cwen

zhen

csev

marquard

zhen

Quel est le nom le plus fréquent?

Quel est le nom le plus fréquent?

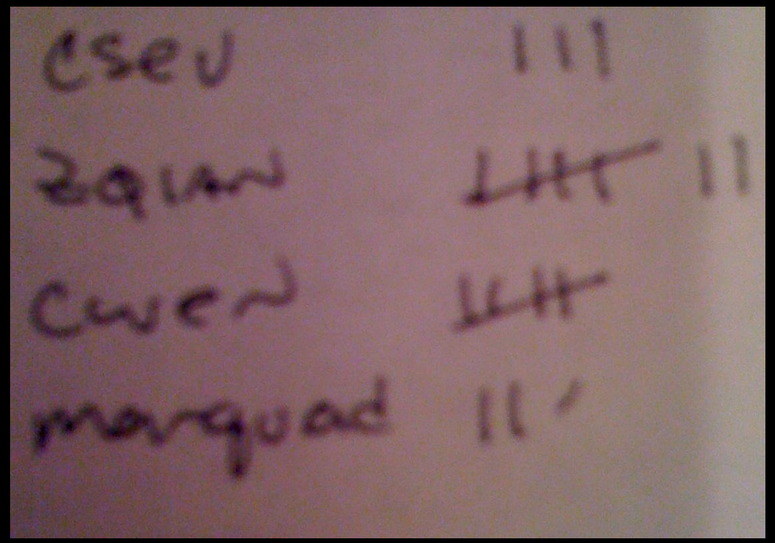
marquard

zhen

csev

zhen

cwen



csev

cwen

zhen

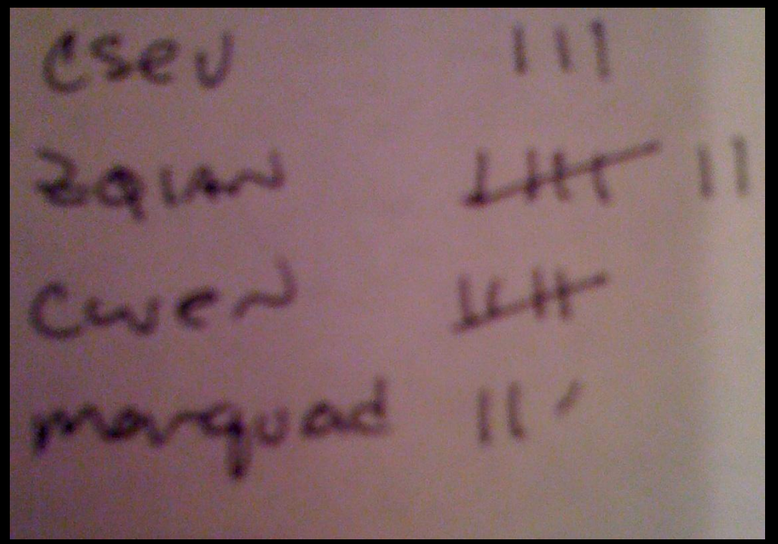
csev

marquard

zhen

Un dictionnaire: beaucoup de compteurs possibles

- Une des utilisations courantes des dictionnaires est celle de compteur qui nous sert à compter le nombre de fois que l'on "voit" quelque chose




```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print ccc
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print ccc
{'csev': 1, 'cwen': 2}
```

Les Tracebacks des Dictionnaires

- C'est une erreur de référencer une clé qui n'est pas dans un dictionnaire
- Nous pouvons utiliser l'opérateur in pour vérifier si une clé est dans le dictionnaire

```
>>> ccc = dict()
>>> print ccc['csev']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> print 'csev' in ccc
False
```

Lorsque nous rencontrons un nouveau nom

- Lorsque nous rencontrons un nouveau nom, nous devons ajouter une nouvelle entrée au dictionnaire et si c'est la deuxième fois ou plus que nous rencontrons le même nom, il nous suffit d'ajouter +1 au compte dans le dictionnaire qui porte le même nom

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian',
'cwen']
for name in names :
    if name not in counts:
       counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print counts
```

{'csev': 2, 'zqian': 1, 'cwen': 2}

La méthode get pour les dictionnaires

- Cette façon de vérifier si une clé est déjà dans un dictionnaire et d'assumer une valeur par défaut s'il n'y a pas de clé est tellement courante, qu'il y a une méthode qui s'appelle la méthode get() qui le fait pour nous

```
if name in counts:
    x = counts[name]
else :
    x = 0
```

```
x = counts.get(name, 0)
```

La valeur clé de défaut if n'existe pas (il n'y aura pas de Traceback).

{'csev': 2, 'zqian': 1, 'cwen': 2}

Comptage simplifié avec get()

- Nous pouvons utiliser get() et fournir une valeur par défaut de zéro lorsqu'une clé n'est pas encore dans le dictionnaire – et ensuite il suffira tout simplement d'ajouter un +1 à cette valeur

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print counts
```

Défaut

{'csev': 2, 'zqian': 1, 'cwen': 2}

Comptage simplifié avec get()

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print counts
```

http://www.youtube.com/watch?v=EHJ9uYx5L58

Écrire des programmes (ou programmer) est une activité très créative et gratifiante. Vous pouvez écrire des programmes pour plusieurs raisons: pour gagner votre vie, pour résoudre un problème d' analyse de données complexes, pour vous amuser ou pour aider quelqu'un d'autre à régler un problème. Ce livre assume que chaque personne à besoin de savoir comment programmer et qu'une fois que vous saurez programmer, vous trouverez ce que vous voulez faire avec vos nouvelles compétences.

Dans notre vie quotidienne nous sommes entourés d'ordinateurs allant des ordinateurs portables aux téléphones portables. Nous pouvons considérer ces ordinateurs comme étant nos "assistants personnels" qui peuvent s'occuper de faire diverses choses à notre place. L'hardware de nos ordinateurs est essentiellement construite afin de nous poser la question, "Et après, que désirez-vous que je fasse?" en continu.

Nos ordinateurs sont rapides, ont une vaste quantité de mémoire et pourraient nous être très utiles si seulement nous connaissions le langage qui nous permettrait de leurs dire ce que voulons qu'ils "fassent après". Si nous pouvions parler ce langage nous pourrions demander aux ordinateurs de faire les taches répétitives à notre place. Il est intéressant de noter que les choses que les ordinateurs sont les meilleurs à sont souvent les choses que les humains trouvent ennuyeuses et abrutissantes.

the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car

La manière de Compter

```
counts = dict()
print ’Entrez une ligne de texte:'
line = raw_input('')

mots = line.split()

print ’Mots:', mots

print ’En train de compter...'
for mot in mots:
    counts[mot] = counts.get(mot,0) + 1
print 'Compte', counts
```

En générale, pour compter les mots d’une ligne de texte, la phrase est fractionnée (split) en mots, puis une analyse de cette phrase (des mots qu’elle contient) est lancée en boucle et un dictionnaire est utilisé afin d’ obtenir le compte de chaque mot individuellement

Compter des mots


```
python wordcount.py
Entrez une ligne de texte:
the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car

Mots: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',
'and', 'the', 'car']
En train de compter…

Compte {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1,
'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7,
'tent': 2}
```

http://www.flickr.com/photos/71502646@N00/2526007974/

```
compte = dict()
print 'Entrez une ligne de texte:'
ligne = raw_input('')
mots = ligne.split()

print 'Mots:', mots
print 'En train de compter...'

for mot in mots:
    compte[mot] = compte.get(mot,0) + 1
print 'Compte', compte
```

```
python wordcount.py
Entrez une ligne de texte:
the clown ran after the car and the car ran
into the tent and the tent fell down on
the clown and the car

Mots: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent',
'and', 'the', 'tent', 'fell', 'down', 'on', 'the',
'clown', 'and', 'the', 'car']
En train de compter...

Compte {'and': 3, 'on': 1, 'ran': 2, 'car': 3,
'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell':
1, 'the': 7, 'tent': 2}
```

Boucles définies et Dictionnaires

- Même si les dictionnaires ne sont pas organisés de façon ordonnée, nous pouvons écrire une bloucle for qui passe par toutes les entrées d'un dictionnaire – en fait la boucle, en recherchant les valeurs, prend en compte toutes les clés du dictionnaire

```
>>> comptes = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for clé in comptes:
...     print clé, comptes[clé]
...
jan 100
chuck 1
fred 42
>>>
```

Récupérer des listes de Clés et de Valeurs

- Vous pouvez obtenir une liste de clés, valeurs, ou d' objets (les deux) à partir d'un dictionnaire

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print liste(jjj)
['jan', 'chuck', 'fred']
>>> print jjj.clés()
['jan', 'chuck', 'fred']
>>> print jjj.valeurs()
[100, 1, 42]
>>> print jjj.objets()
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

Qu'est-ce qu'un 'tuple'? – arrive bientôt...

Bonus: Deux Variables d'Itération!

- Nous passons en boucle à travers une paire de clé-valeur dans un dictionnaire en utilisant *deux* variables d'itération
- Pour chaque itération, la première variable est la clé et la seconde est la valeur *correspondante* à la clé

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42,
'jan': 100}
>>> for aaa,bbb in jjj.items() :
...           print aaa, bbb
...
jan 100
chuck 1
fred 42
>>>
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

```
nom = raw_input('Entrer un fichier:')
handle = open(nom,'r')
texte = handle.read()
mots = text.split()

comptes = dict()
for mot in mots:
    comptes[mot] = comptes.get(mot,0) + 1

bigcount = None
bigword = None
for mot,compte in comptes.objets():
    if bigcount is None or compte > bigcount:
        bigword = mot
        bigcount = compte

print bigword, bigcount
```

```
python mots.py
Entrer un fichier: words.txt
to 16
```

```
python mots.py
Entrer un fichier: clown.txt
the 7
```

Résumé

- Qu'est-ce qu'une collection?
- Les Listes versus les Dictionnaires
- Les constantes de dictionnaires
- Le mot le plus fréquent
- L'utilisation de la méthode get()
- Le fractionnement et le manque d'ordre
- Écrire des boucles de dictionnaires
- Coup d'oeil rapide sur les Tuples
- Réorganiser les dictionnaires

Remerciements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Translation: Stephanie Kamidian

...