

Initiation à la programmation en Python

Damien Vergnaud

École Normale Supérieure

25 mars 2015

Les dictionnaires

- Les **listes** sont des suites ordonnées d'éléments.
Il est facile d'accéder à un élément quelconque ... à condition expresse de connaître son emplacement ...
- Les **dictionnaires** constituent un autre type composite.
- Les éléments enregistrés ne sont pas disposés dans un ordre immuable
- Ils sont accessibles à l'aide d'un index spécifique que l'on appellera une **clé**, laquelle pourra être alphabétique, numérique, ou même d'un type composite sous certaines conditions.

Création d'un dictionnaire

- Les éléments d'un dictionnaire sont enfermés dans une paire d'accolades.
- Un dictionnaire vide sera donc noté `{}`

```
>>> dico = {}
>>> dico['computer'] = 'ordinateur'
>>> dico['mouse'] = 'souris'
>>> dico['keyboard'] = 'clavier'
>>> print dico
{'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

- Les index s'appellent des **clés**, et les éléments peuvent donc s'appeler des paires **clé-valeur**.
- Nous utilisons les clés pour extraire une valeur d'un dictionnaire :

```
>>> print dico['mouse']
souris
```

Opérations sur les dictionnaires

- Pour ajouter de nouveaux éléments à un dictionnaire, il suffit de créer une nouvelle paire clé-valeur.
- Pour enlever des éléments, il faut utiliser l'instruction `del` :

```
>>> invent = {'pommes': 430, 'bananes': 312, 'oranges' : 274, \
              'poires' : 137}
>>> print invent
{'oranges': 274, 'pommes': 430, 'bananes': 312, 'poires': 137}
>>> del invent['pommes']
>>> print invent
{'oranges': 274, 'bananes': 312, 'poires': 137}
```

- La fonction `len` renvoie le nombre d'éléments d'un dictionnaire

Les dictionnaires sont des objets

- La méthode `keys()` renvoie la liste des clés utilisées dans le dictionnaire :

```
>>> print dico.keys()
['computer', 'keyboard', 'mouse']
```

- La méthode `values()` renvoie la liste des valeurs mémorisées dans le dictionnaire :

```
>>> print invent.values()
[274, 312, 137]
```

- La méthode `has_key()` permet de savoir si un dictionnaire comprend une clé déterminée :

```
>>> print invent.has_key('bananes')
1
>>> if invent.has_key('pommes'):
    print 'nous avons des pommes'
else:
    print 'pas de pommes'

pas de pommes
```

Les dictionnaires sont des objets

- La méthode `copy()` permet d'effectuer une vraie copie d'un dictionnaire.

```
>>> stock = invent
>>> print stock
{'oranges': 274, 'bananes': 312, 'poires': 137}

>>> del invent['bananes']
>>> print stock
{'oranges': 274, 'poires': 137}
```

```
>>> magasin = stock.copy()
>>> magasin['prunes'] = 561
>>> print magasin
{'oranges': 274, 'prunes': 561, 'poires': 137}
>>> print stock
{'oranges': 274, 'poires': 137}
>>> print invent
{'oranges': 274, 'poires': 137}
```

Parcours d'un dictionnaire

- Une boucle `for` permet de traiter successivement tous les éléments contenus dans un dictionnaire
- Ce sont les clés du dictionnaire qui seront successivement affectées à la variable de travail, et non les valeurs.

```
>>> invent ={"oranges":274, "poires":137, "bananes":312}
>>> for clef in invent:
... print clef
```

```
poires
bananes
oranges
```

```
>>> for clef in invent:
... print clef, invent[clef]
```

```
poires 137
bananes 312
oranges 274
```

Expressions régulières

- Le module `re` permet d'utiliser des expressions régulières au sein de Python.

```
import re
```

- Les expressions régulières sont de très puissants outils de manipulation de texte et de données.
- La fonction `search` permet de rechercher un motif au sein d'une chaîne de caractères :

```
>>> import re
>>> chaine = "abcdef"
>>> if re.search("bcd",chaine):
    print chaine
abcdef
```

Métacaractères

- . n'importe quel caractère
- ^ correspond au début d'une chaîne
- \$ correspond à la fin d'une chaîne
- * correspond à 0 ou plusieurs occurrences de l'expression qui précède
- + correspond à 1 ou plusieurs occurrences de l'expression qui précède
- ? correspond à 0 ou 1 occurrence de l'expression ou caractère qui précède
- [ABC] reconnaît A ou B ou C
- [A-Z] reconnaît n'importe quel lettre
- [^AB] reconnaît n'importe quel caractère sauf A ou B
- (ABC) reconnaît ABC
- {n,m} reconnaît de n à m fois l'expression qui précède
- {n,} reconnaît au moins n fois l'expression qui précède

Exemples

<code>^L</code>	la lettre L au début de la chaîne
<code>[A-Z]\$</code>	n'importe quelle lettre à la fin de la chaîne
<code>(toto)</code>	le motif toto
<code>(toto).*(tutu)</code>	toto, puis n'importe quel caractère, puis tutu
<code>(toto tutu)</code>	motif toto ou tutu
<code>(tat)[^u]</code>	motif tat puis n'importe quel caractère sauf le u
<code>(toto)2,4</code>	motif toto répété de 2 à 4 fois

Autres métacaractères

`\d` un chiffre (0-9)

`\D` tout sauf un chiffre

`\s` une espace (ou tabulation ou saut de ligne)

`\S` tout sauf un espace

`\w` une lettre ou un chiffre

`\W` tout sauf une lettre ou un chiffre

Expressions régulières : groupes

- Les parenthèses regroupent un élément d'un motif

```
texte='une poouule sur un mur')
resultat=re.search('(po*uule).*un\s(.*)',texte)

print resultat.group(0) # 'poouule sur un mur'
print resultat.group(1) # 'poouule'
print resultat.group(2) # 'mur'
print resultat.groups() # ('poouule','mur')
```

Expressions régulières : autres méthodes

- `split`

```
>>> re.split('\W+', 'Words, words, words.')
['Words', 'words', 'words', '']
>>> re.split('(\W+)', 'Words, words, words.')
['Words', '', ' ', 'words', ' ', ' ', 'words', '.', '']
>>> re.split('\W+', 'Words, words, words.', 1)
['Words', 'words, words.']
```

- `sub`

```
>>> re.sub('[abc]', 'o', 'Mark')
'Mork'
>>> re.sub('[abc]', 'o', 'rock')
'rook'
>>> re.sub('[abc]', 'o', 'caps')
'oops'
```

- ...et beaucoup d'autres :

<http://docs.python.org/library/re.html>