Algo & Prog, avec Python
(L1-Sciences)
TD n° 8, Automne 2016

Ensembles

Exercice 8.1 a) Définissez l'ensemble `alphabet` contenant les 26 lettres minuscules de l'alphabet.
b) Définissez une fonction `nb_minuscules(s)` prenant une chaîne de caractères `s`, et retournant le nombre de lettres minuscules de `s`, en testant l'appartenance de chaque caractère de `s` à l'ensemble alphabet.

Exercice 8.2 Définissez un ensemble `e` de 20 entiers aléatoires de `[0,40]`.

Dictionnaires

Exercice 8.3 Créez un petit dictionnaire `d1` ayant trois couples `var:val`.
a) Faites afficher le type de `d1`.
b) Faites afficher la liste des clés de `d1`.
c) Faites afficher l'ensemble des valeurs de `d1`.

Exercice 8.4 Programmez une fonction `the_keys(d,v)` retournant l'ensemble des clés d'un dictionnaire quelconque `d`, dont la valeur associée est `v`. *Rappel : les clés sont uniques, mais pas les valeurs !* Exemple :

```
the_keys({'a':5, 'b':6, 'c':5, 'd':4}, 5) → {'a', 'c'}
```

Exercice 8.5 On considère la suite (u_n) définie par récurrence de la manière suivante :

$$u_0 = 2 \quad \text{et} \quad u_n = \frac{1}{2}u_{n-1} + 3$$

a) Programmez la fonction `u(n)` par récurrence.
b) Reprogrammez-la par une boucle.
c) Faites afficher la liste $[u_0, u_1, u_2, \cdots, u_N]$ en utilisant la fonction `u`. Quel est le coût du calcul de cette liste en fonction de `N` ?
d) Reprogrammez la fonction `u(n)` sous la forme d'une **mémo-fonction**, qui se souvienne des calculs qu'elle a déjà effectués. Vous nommerez `mem` la variable globale qui représente la table de stockage des calculs déjà effectués.
e) Le coût du calcul de la liste $[u_0, u_1, u_2, \cdots, u_N]$ a-t-il changé ?

Exercice 8.6 a) Programmez une fonction `random_dict()`, retournant un dictionnaire aléatoire de longueur `10`. Les `10` clés distinctes seront choisies au hasard parmi les entiers de `[0,20]`. Les valeurs associées à ces clés [non nécessairement distinctes] seront choisies parmi les voyelles `'a','e','i','o','u','y'`. Posons ensuite `d2 = random_dict()`.
b) Faites afficher un avertissement si la longueur de `d2` n'est pas égale à `10`.
c) Faites afficher le nombre de clés paires de `d2`.
d) Faites afficher les valeurs distinctes de `d2`.

Algo & Prog, avec Python
(L1-Sciences)
TP n° 8, Automne 2016

Ensembles

Exercice 8.1 Créez un petit ensemble `e1` ayant trois éléments de types distincts. Demandez quel est le type de `e1`.

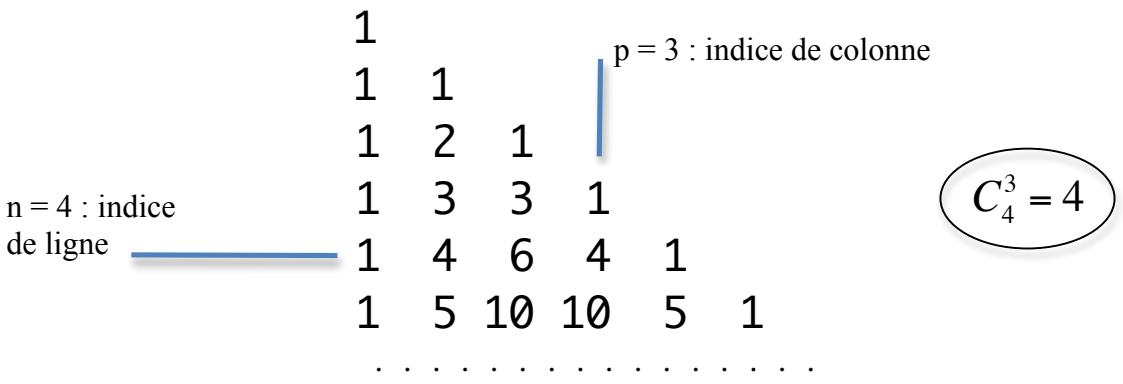
Exercice 8.2 a) Définissez deux ensembles `e2` et `e3` contenant chacun 10 entiers aléatoires de `[0,20]`.
b) Faites afficher la réunion `e2` ∪ `e3`, l'intersection `e2` ∩ `e3` et la différence `e2 - e3`.

Exercice 8.3 Histoire de s'amuser (!), supposons que Python ne dispose pas de la classe `set`, mais seulement de la classe `list`. Comment programmeriez-vous les fonctions `difference(L1,L2)`, `reunion(L1,L2)`, et `intersection(L1,L2)` sur des listes d'entiers non triées représentant des *ensembles non ordonnés* ?

Dictionnaires

Exercice 8.4 a) Créez un petit dictionnaire `d1` ayant trois couples `var:val`. Demandez quel est le **type** (⇔ la **classe**) de `d1`.
b) En utilisant `type`, demandez si `d1` est bien dans la classe obtenue en a). Demandez quelle est la valeur de la variable `int`.

Préliminaire à l'exercice 8.5: vous connaissez tous le Triangle de Pascal [sinon : Google !]. Il est bâti sur les coefficients C_n^p - aussi notés $\binom{n}{p}$ - et se construit par des additions :



Autrement dit :

$$C_n^p = C_{n-1}^p + C_{n-1}^{p-1} \qquad (10 = 4 + 6 = 6 + 4)$$

Nous nous proposons de programmer la fonction `binomial(n,p)` retournant C_n^p.

Exercice 8.5 a) Vérifiez que la fonction suivante conduit à une pauvre programmation :

```
def binomial(n,p) :
    if p == 0 or n == p :      # les bords du triangle
        return 1
    else :
        return binomial(n-1,p) + binomial(n-1,p-1)
```

Demandez par exemple à calculer C_5^3, puis C_{52}^{13} qui représente le nombre de mains de 13 cartes parmi 52 au bridge.

b) Programmez `binomial` sous la forme d'une **mémo-fonction** qui se souvient des calculs déjà effectués [stockés dans un dictionnaire global `mem`]. Vérifiez qu'alors le calcul de C_{52}^{13} est immédiat !

c) Avec une double boucle `for`, faites afficher le Triangle de Pascal jusqu'à la ligne 6 incluse. Tâchez d'avoir des colonnes bien alignées !…

N.B. Il existe une autre formule pour C_n^p, basée sur des factorielles et utilisée en combinatoire :

$$C_n^p = \frac{n!}{p!(n-p)!}$$

Vous pouvez la programmer en Python, en vous posant quand même la question : ne suis-je pas en train de faire trop de multiplications ?...

Exercice complémentaire

Dans le TD5, vous avez fait connaissance avec les méthodes `index` et `count` sur les séquences, en particulier sur les chaînes de caractères. Vous êtes aujourd'hui invités à essayer la méthode `split` de la classe `str` qui récupère la liste des mots d'une phrase séparés par `sep` :

```
>>> 'le chien et le loup font la course'.split()     # sep = ' ' par défaut
['le', 'chien', 'et', 'le', 'loup', 'font', 'la', 'course']
>>> 'Nice, Antibes, Monaco'.split(', ')              # sep = ', '
['Nice', 'Antibes', 'Monaco']
```

Exercice 8.6 a) Utilisez les méthodes `split` et `count` pour programmer une fonction `frequences(s)` prenant une chaîne `s` et retournant un dictionnaire contenant les fréquences d'apparition de chaque mot de `s`. Exemple :

```
>>> frequences('do re do mi do la la mi la')
{'do':3, 'la':3, 'mi':2, 're':1}
```

b) Modifiez votre solution pour que le résultat soit une liste de couples triée par la fréquence :

```
>>> frequences('do re do mi do la la mi la')
[('do',3), ('la',3), ('mi',2), ('re',1)]
```

c) En déduire une fonction `plus-frequents(s)` retournant les mots les plus fréquents :

```
>>> plus_frequents('do re do mi do la la mi la')
['la', 'do']
```

N.B. Cet exercice illustre vraiment le « style Python »...