TD 3 - Programmation python Utilisation des listes

Informatique
MPSI - Lycée Thiers

2014/2015

Exercice 1

1. Ecrire deux fonctions de paramètre `N` entier naturel, et qui retournent la liste des `N` premiers termes de la suite arithmétique de raison 12 et de premier terme 1 :

$$\forall n \in \mathbb{N},\ u_n = 12n + 1$$

par deux méthodes différentes :

   1.1 A l'aide d'une boucle.
   1.2 Par compréhension de liste.

2. Comparer les temps d'exécution des deux méthodes à l'aide de la fonction `clock()` du module `time`, et du petit script suivant :

```
from time import clock
def duree(fonction, n=1000000):# durée d'exécution de fonction(n)
    debut = clock()          # temps début
    fonction(n)              # appel de fonction(n)
    fin = clock()              # temps fin
    return fin - debut          # durée retournée en seconde
```

A l'aide d'une boucle :

```
def suite1(n):
    result = []
    for i in range(n):
        result.append(12*i+1)
    return result
```

Par compréhension de liste :

```
def suite2(n):
    return [12*i+1 for i in range(n)]
```

La deuxième méthode est deux fois plus rapide.

```
>>> duree(suite1)
0.2178670000000003
>>> duree(suite2,1000000)
0.1068560000000006
```

Lorsque `n` varie ce rapport demeure identique, leur temps d'exécution est du même ordre de grandeur.

Exercice 2

1. Ecrire une fonction `moyenne()` prenant en paramètre une liste de nombres et qui retourne sa moyenne.
2. Ecrire une fonction `variance()` prenant en paramètre une liste de nombres et qui retourne sa variance.
3. On pourra les tester sur des listes de nombres aléatoires que l'on aura créées à l'aide de la fonction `random()` du module `random` qui renvoie un décimal (`float`) aléatoire dans $[0, 1[$.

1) Réponse :

```
def moyenne(liste):
    if (liste != []):
        return sum(liste) / len(liste)
```

```
>>> from random import random
>>> L=[random() for i in range(100)]
>>> moyenne(L)
0.4940795186378354 5
```

Calcul de la variance : à l'aide de la définition $V(X) = E((X - E(X))^2)$.

```
def variance(liste):
    if (liste != []):
        Esp = moyenne(liste)
        liste2 = [(x-Esp)**2 for x in liste]
        return moyenne(liste2)
```

A l'aide de la formule de König-Huygens $V(X) = E(X^2) - E(X)^2$.

```
def variance(X):
    return moyenne([x**2 for x in X]) - moyenne(X)**2
```

Exercice 3

Construire par compréhension de liste les listes suivantes :

1. La liste des logarithmes népériens (ln) des entiers compris entre 1 et 20.
2. La liste des entiers compris entre 0 et 100 qui sont pairs et qui ne sont pas des multiples de 3.
3. La liste `['@-@', '@--@', '@---@', '@----@', '@-----@']`.
4. La liste des couples d'entiers entre 0 et 10 dont la somme est un multiple de 5.
5. Ecrire une fonction qui prend pour paramètre un entier `n` et retourne sous forme d'une liste de listes la matrice identité de rang `n`.

1) Réponse : liste des $\ln(n)$ pour n entier compris entre 1 et 20.

```
>>> from math import log    # import de log du module math
>>> liste1 = [log(n) for n in range(1,21)]
>>> print(l)
[0.0, 0.693147180559453, 1.0986122886681098, 1.386294361119890 6,
1.6094379124341003, 1.791759469228055, 1.945910149055313 2,
2.0794415416798357, 2.1972245773362196, 2.302585092994046,
2.3978952727983707, 2.484906649788000 4, 2.5649493574615367,
2.6390573296152584, 2.70805020110221, 2.772588722239781,
2.833213344056216, 2.8903717578961645, 2.944438979166440 3,
2.995732273553991]
```

Remarquer que ça croît très lentement...

2) Réponse : liste des entiers pairs et non multiples de 3 entre 0 et 100.

```
>>> liste2 = [n for n in range(0,101) if (n % 2 == 0) and (n % 3 != 0)]
>>> print(liste2)
[2, 4, 8, 10, 14, 16, 20, 22, 26, 28, 32, 34, 38, 40, 44, 46, 50, 52, 56, 58, 62,
64, 68, 70, 74, 76, 80, 82, 86, 88, 92, 94, 98, 100]
>>> liste2 = [n for n in range(0,101,2) if (n % 3 != 0)]
>>> print(liste2)
[2, 4, 8, 10, 14, 16, 20, 22, 26, 28, 32, 34, 38, 40, 44, 46, 50, 52, 56, 58, 62,
64, 68, 70, 74, 76, 80, 82, 86, 88, 92, 94, 98, 100]
```

Une alternative, plus rapide.

3) Réponse :

```
>>> ['@' + '-' * n + '@' for n in range(1,6)]
['@-@', '@--@', '@---@', '@----@', '@-----@']
```

4) Réponse :

```
>>> [(i, j) for i in range(11) for j in range(11) if (i+j) % 5 == 0]
[(0, 0), (0, 5), (0, 10), (1, 4), (1, 9), (2, 3), (2, 8), (3, 2), (3, 7), (4, 1),
(4, 6), (5, 0), (5, 5), (5, 10), (6, 4), (6, 9), (7, 3), (7, 8), (8, 2), (8, 7),
(9, 1), (9, 6), (10, 0), (10, 5), (10, 10)]
```

5) Réponse :

```
def IdMat(n):
    return [[0 for j in range(k)]+[1]+[0 for j in range(n-1-k)] for k in range(n)]
```

```
>>> IdMat(2)
[[1, 0], [0, 1]]
>>> IdMat(3)
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
>>> IdMat(4)
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]
```

Exercice 4 : usage de la table ASCII

Les caractères non accentués, chiffres et autres symbôles accessibles au clavier ou non, sont disponibles dans la table ASCII de caractères, numérotés de 0 à 255 (0xFF).
La fonction `chr(n)` retourne le caractère de la table de code ASCII `n`.
La fonction `ord()` retourne le code ASCII d'un caractère dans la table.
(les caractères de 'a' à 'z' sont rangés dans l'ordre alphabétique de 97 à 122, les caractères de 'A' à 'Z' de 65 à 90.)

1. Ecrire un script pour afficher à l'écran les caractères de la table ASCII de code allant de 33 à 126. On les présentera par ligne de 16 caractères.
2. Ecrire une fonction `maj()` qui prend en argument une chaine de caractère non accentuée et qui renvoie la chaîne avec minuscules converties en majuscules.

1) Solution :

```
>>> c=''
>>> for i in range(33,127):
...     c = c + str(i) + ':' + chr(i) + ' '
...     if (i-32) % 16 == 0: c = c + '\n'
...
>>> print c
33:! 34:¨ 35:# 36:$ 37:% 38:& 39:' 40:( 41:) 42:* 43:+ 44:, 45:- 46:. 47:/ 48:0
49:1 50:2 51:3 52:4 53:5 54:6 55:7 56:8 57:9 58:: 59:; 60:< 61:= 62:> 63:? 64:@
65:A 66:B 67:C 68:D 69:E 70:F 71:G 72:H 73:I 74:J 75:K 76:L 77:M 78:N 79:O 80:P
81:Q 82:R 83:S 84:T 85:U 86:V 87:W 88:X 89:Y 90:Z 91:[ 92:\93:] 94:^ 95:_ 96:‘
97:a 98:b 99:c 100:d 101:e 102:f 103:g 104:h 105:i 106:j 107:k 108:l 109:m 110:n
111:o 112:p 113:q 114:r 115:s 116:t 117:u 118:v 119:w 120:x 121:y 122:z 123:{
124:| 125:} 126:~
```

2) Solution :

```
>>> def maj(chaine):
...     result = ''
...     for i in range(len(chaine)):
...         code = ord(chaine[i])
...         if (code >= ord('a')) and (code <= ord('z')):
...             result = result + chr(code + ord('A') - ord('a'))
...          else :
...             result = result + chr(code)
...      return result
...
>>> print(maj('bonjour !'))
BONJOUR !
```

Exercice 5 : le chiffrement de César

C'est une méthode très simple de chiffrement de messages en un texte crypté pour le rendre "illisible" à qui n'en a pas la clef. Dans une message en lettres majuscules supprimer tous les espaces et symbôles, puis changer chaque lettre par sa *N*ième lettre suivante dans l'ordre alphabétique (après z on reprend en a), où N est un entier entre 1 et 25.

1. Ecrire une fontion CesarCrypt() qui prend deux paramètres, la clef de chiffrement *N* et une chaine de caractère en majuscule sans symbole ni espace, et renvoie le chiffrement de César de la chaine.
2. Ecrire une fonction CesarDecrypt() qui prend deux paramètres, un texte crypté et la clef, et retourne le texte décrypté.

1) **Solution :** Fonction de cryptage

```
>>> def CesarCrypt(ch,n):
...     result = ''
...     for i in range(len(ch)):
...     result = result + chr((ord(ch[i]) + n-ord('A')) % 26 + ord('A'))
...     return result
...
>>> print(CesarCrypt('BONJOUR',1))
CPOKPVS
```

2) **Solution :** Fonction de décryptage

```
>>> def CesarDecrypt(ch,n):
...     return CesarCrypt(ch, -n)
...
>>> CesarDecrypt('CPOKPVS',1)
'BONJOUR'
```