Algorithmique et Programmation 1 – TD - TP 10

Compréhensions de listes

CORRECTION

Exercice 1 - Compréhensions de listes (1)

Quelles sont les listes construites par les expressions suivantes ?

```
1 [2 * k for k in range(-2,3)]
2 [k * (k+1) / 2 for k in range(2,5)]
3 [1 for c in 'abc']
4 [c+c for c in 'abc']
5 [(1,2,k) for k in range(1,3)]
6 [[i, j] for i in range(0, 3) for j in range(3, 5)]
7 [k for k in range(10) if k % 3 == 0]
8 [k + 2 if k <= 5 else k * 2 for k in range(10)]
9 [(i, j) for i in range(0, 4) for j in range(2, 6) if (i + j)%2 == 0 ]
```

```
>>> [2 * k for k in range(-2,3)]
[-4, -2, 0, 2, 4]
>>> [k * (k+1) / 2 for k in range(2,5)]
[3.0, 6.0, 10.0]
>>> [1 for c in 'abc']
[1, 1, 1]
>>> [c+c for c in 'abc']
['aa', 'bb', 'cc']
>>> [(1,2,k) for k in range(1,3)]
[(1, 2, 1), (1, 2, 2)]
>>> [[i, j] for i in range(0, 3) for j in range(3, 5)]
[[0, 3], [0, 4], [1, 3], [1, 4], [2, 3], [2, 4]]
>>> [k for k in range(10) if k % 3 == 0]
[0, 3, 6, 9]
>>> [k + 2 if k <= 5 else k * 2 for k in range(10)]
[2, 3, 4, 5, 6, 7, 12, 14, 16, 18]
>>> [(i, j) for i in range(0, 4) for j in range(2, 6) if (i + j)%2 == 0 ]
[(0, 2), (0, 4), (1, 3), (1, 5), (2, 2), (2, 4), (3, 3), (3, 5)]
```

Exercice 2 - Compréhensions de listes (2)

A l'aide d'une compréhension :

1. Construire la liste des 5 premiers entiers pairs

```
[n for n in range(0, 9, 2)]

# ou

[n for n in range(0, 9) if n % 2 == 0]
```

2. Construire la liste de tous les couples (i, i + 1) pour i compris entre 1 et 10

```
[(i, i+1) for i in range(1, 11)]
```

3. Donner une définition de la fonction qui, étant donnée une liste d'entiers `l`, renvoie la liste des entiers pairs de `l`

```
def liste_pairs(liste) :
    """List --> List.
    Retourne la sous-liste des entiers pairs de la liste donnée en paramètre"""

    return [elem for elem in liste if elem % 2 == 0]
```

4. Donner une définition de la fonction qui, étant donnée une chaîne de caractères `s`, renvoie la liste des voyelles contenues dans `s`

```
def liste_voyelles(chaine) :
    """Str --> List
    Retourne la liste des voyelles contenues dans la chaine de caractères"""

    return [c for c in chaine if c in 'aeiouyAEIOUY']
```

5. Donner une définition de la fonction qui, étant donnée une liste de liste `l`, renvoie la liste contenant les nombres entiers multipliés par 2 et les chaînes de caractères multipliées par 3. Par exemple :

```
>>> l = [[0, 'a'], [2, 'b'], [3, 'c']]
>>> double_int_triple_string(l)
[0, 'aaa', 4, 'bbb', 6, 'ccc']
```

```
def double_int_triple_string(liste) :
    """List --> List
    Retourne la liste composés des éléments de la liste donnée en paramètre, dont
    les entiers ont été multipliés par 2 et les chaines de caractères par 3"""

    return [elem*2 if type(elem) == int else elem*3 for l in liste for elem in l]
```

Exercice 3 - Crible d'Eratosthène

Cet exercice consiste à implémenter le crible d'Eratosthène qui décrit un moyen de calculer la listes des nombres premiers inférieurs à un entier fixé. On rappelle qu'un nombre est premier s'il n'est divisible que par 1 et lui-même. On rappelle également que, par convention, 1 n'est pas un nombre premier.

La méthode du crible d'Eratosthène consiste à créer dans un premier temps la liste des entiers compris entre 2 et n, puis itérer le processus suivant :

1. Récupérer le premier élément de la liste (ce nombre est un nombre premier)
2. Retirer de la liste restante tous les multiples de ce nombre
3. Recommencer à l'étape 1. tant qu'il reste des éléments dans la liste

Par exemple, pour $n = 10$, on commence par créer la liste `entiers = [2, 3, 4, 5, 6, 7, 8, 9, 10]`, la liste contenant les nombres premiers est initialement vide `premiers = []`

- `entiers[1] = 2` est un nombre premier. On retire de la liste `entiers` tous les multiples de 2. On obtient `entiers = [3, 5, 7, 9]`, `premiers = [2]`
- `entiers[1] = 3` est un nombre premier. On retire de la liste `entiers` tous les multiples de 3. On obtient `entiers = [5, 7]`, `premiers = [2, 3]`
- `entiers[1] = 5` est un nombre premier. On retire de la liste `entiers` tous les multiples de 5. On obtient `entiers = [7]`, `premiers = [2, 3, 5]`
- `entiers[1] = 7` est un nombre premier. On retire de la liste `entiers` tous les multiples de 7. On obtient `entiers = []`, `premiers = [2, 3, 5, 7]`

- La liste est vide. La liste des nombres premiers inférieurs ou égaux à 10 est donc `premiers = [2, 3, 5, 7]`

1. A l'aide d'une compréhension, définir une fonction `liste_non_multiple` qui, étant donné un entier n non nul et une liste d'entiers `l`, renvoie la liste des éléments de `l` qui ne sont pas multiples de n. Par exemple :

```
>>> liste_non_multiple(2,[2,3,4,5,6,7,8,9,10])
[3, 5, 7, 9]
```

```
def liste_non_multiple(n, l) :
    """Int x List --> List, n > 0
    Retourne la liste des éléments de l qui ne sont pas des multiples de n"""

    assert n > 0

    return [elem for elem in l if elem % n != 0]
```

2. Donner la définition et la spécification de la fonction `eratosthene` qui calcule la liste des nombres premiers inférieurs ou égaux à un entier n (supérieur ou égal à 2) en utilisant le crible d'Eratosthène décrit ci-dessus.

```
def eratosthene(n) :
    """Int --> list
    Retourne la liste des nombres premiers inférieurs ou égaux à n"""

    premier = []
    reste = [i for i in range(2, n+1)]

    while len(reste) > 0 :
        premier.append(reste[0])
        reste = liste_non_multiple(reste[0], reste)
    return premier
```

3. Un facteur premier d'un entier n est un nombre premier qui divise n. A l'aide de la fonction `eratosthene`, et en utilisant une compréhension, donner la définition et la spécification de la fonction `liste_facteurs_premiers` qui, étant donné un entier n supérieur ou égal à 2, calcule la liste des facteurs premiers de n, c'est à dire la liste des nombres premiers inférieurs ou égaux à n et qui divisent n. Par exemple :

```
>>> liste_facteurs_premiers(2)
[2]
>>> liste_facteurs_premiers(10)
[2, 5]
>>> liste_facteurs_premiers(2*3*4*7*9)
[2, 3, 7]
```

```
def liste_facteurs_premiers(n) :
    """Int --> List
    Retourne la liste des facteurs premiers de n"""

    return [i for i in range(n+1) if i in eratosthene(n) and n % i == 0]
```

Exercice 4 - Triplets

1. Donner , en utilisant une compréhension, une définition de la fonction `triplets` qui retourne la liste des triplets (i,j,k) sur l'intervalle $[1,n]$ (avec n un entier naturel). Par exemple :

```
>>> triplets(2)
[(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1, 1), (2, 1, 2), (2, 2, 1), (2, 2, 2)]
```

```
def triplets(n) :
    """Int --> List
    Retourne la liste des triplets (i,j,k) sur l'intervalle [1, n]"""

    return [(i, j, k) for i in range(1, n+1)
                      for j in range(1, n+1)
                      for k in range(1, n+1)]
```

2. Donner, en utilisant une compréhension, une définition de la fonction `décomposition` qui retourne la liste des triplets (i,j,k) sur l'intervalle $[1,n]$ (avec n un entier naturel) tels que $i+j=k$. Par exemple :

```
>>> decomposition(2)
[(1, 1, 2)]
>>> decomposition(3)
[(1, 1, 2), (1, 2, 3), (2, 1, 3)]
```

```
def decomposition(n) :
    """Int --> list
    Retourne la liste des triplets (i,j,k) sur l'intervalle [1, n] tels que i + j = k"""

    return [(i, j, k) for i in range(1, n+1)
                      for j in range(1, n+1)
                      for k in range(1, n+1) if i + j == k]
```

3. Donner, en utilisant une compréhension, une définition de la fonction `differents` qui retourne la liste des triplets (i,j,k) sur l'intervalle $[1,n]$ (avec n un entier naturel) tels que i, j et k sont tous différents. Par exemple :

```
>>> differents(2)
[]
>>> differents(3)
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

```
def differents(n) :
    """Int --> List
    Retourne la liste des triplets (i,j,k) sur l'intervalle [1, n]
    tels que i, j et k sont différents"""

    return [(i, j, k) for i in range(1, n+1)
                      for j in range(1, n+1)
                      for k in range(1, n+1) if i != j and i != k and j != k]
```