

Première NSI - Algorithmie

Travaux dirigés : tableaux à deux dimensions, liste et dictionnaires par compréhension

qkzk

2020/07/06

Tableaux à deux dimensions

1. Coordonnées dans une grille

On considère une grille rectangulaire représentée en mémoire par un tableau à deux dimension :

```
grille = [[1, 2, 1, 5],  
          [2, 1, 4, 3],  
          [1, 4, 5, 1]]
```

Chaque élément de la grille est un entier.

L'algorithme, incomplet, qui suit permet de cumuler les éléments de la grille :

```
somme = 0  
pour chaque ligne de la grille faire :  
    pour chaque cellule d'une ligne faire :  
        COMPLÉTER  
    fin du pour  
fin du pour
```

1. Compléter l'algorithme
2. Combien d'additions sont réalisées lorsqu'on applique l'algorithme à la grille donnée en exemple ? Pour une grille contenant L lignes et C colonnes ?
3. Traduire cet algorithme dans une fonction python `cumul_grille`

2. Élément extrême

On a relevé dans un tableau appelé `temperatures` les températures à midi de chaque jour de la semaine.

Exemple : `temperatures = [12, 10, 14, 11, 13, 16, 12]`

Ces relevés hebdomadaires ont ensuite été regroupés dans un tableau `releves`

Exemple :

```
releves = [[12, 10, 14, 11, 13, 16, 12],  
          [10, 13, 15, 10, 11, 11, 15],  
          ...  
          [11, 12, 13, 12, 14, 11, 14]]
```

On souhaite connaître le numéro de la semaine et le jour de la semaine durant lequel la température a été maximale, minimale.

Voici la proposition de Jérôme :

```
temp_max = 0  
for semaine in releves:  
    for temperature in semaine:  
        if temperature > temp_max:  
            temp_max = temperature  
return temperature
```

Lorsqu'il exécute son script, Jérôme obtient une erreur :

```
syntax error: 'return' outside a function
```

1. D'après le message d'erreur, quel est le problème ? Rectifiez le code afin qu'il ne lève plus cette erreur.
2. L'algorithme de Jérôme répond-il au problème ? Quelle sera l'information obtenue ?
3. Proposez une correction du programme afin d'obtenir l'information souhaitée .

3. Modifier un tableau de pixels

Le programme suivant génère une image rectangulaire comportant un dégradé du blanc au noir :

```
degradé
from PIL import Image

def degrade_blanc_noir():
    # nouvelle image, 255 de large, 128 de haut
    img = Image.new('RGB', (255, 128))
    # on charge la matrice des pixels
    pixels = img.load()

    for x in range(255):
        for y in range(128):
            # attention à la notation [x, y] !!!
            pixels[x, y] = (255 - x, 255 - x, 255 - x)

    img.show() # afficher dans la console
```

1. Adaptez le code de cette fonction pour générer un dégradé du noir au blanc.
2. On souhaite tourner l'image afin de produire un dégradé vertical. Adaptez la fonction.

degradé vertical

4. Matrice enregistrée en ligne

On a enregistré le nombre de naissances dans une petite maternité pour chaque jour de la semaine. Certains langages ne permettent pas de construire des listes de listes. À la suite d'un enregistrement en mémoire, le tableau ci-dessous a été enregistré dans une longue liste Python :

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
1	1	3	2	3	2
2	2	4	2	6	4
3	1	3	2	1	4

```
naissances = [1, 3, 2, 3, 2, 2, 4, 2, 6, 4, 1, 3, 2, 1, 4]
```

Lorsqu'on manipule de telles données, il est courant d'utiliser 2 indices :

```
for i in range(3):
    for j in range(5):
        jour = 5 * i + j
        nb_naissance = naissances[jour]
        print(i, j, nb_naissance)
```

1. Faire tourner le programme précédent à la main et écrire les affichages qu'il produit.
2. On souhaite créer un programme qui affiche, pour n'importe quelle liste Python de longueur $n \times p$ ses éléments par bloc de taille p .

Le programme précédent est un exemple pour des blocs de taille 3×5 .

Proposer une fonction qui :

- prend en entrée une liste de taille $n \times p$ et les nombres entiers n et p ;
- ne retourne rien ;
- affiche les éléments par bloc ainsi :

```
1 3 2 3 2
2 4 2 6 4
1 3 2 1 4
```

On utilisera le paramètre *positionnel* de la fonction `print` :

```
>>> for x in range(3):
...     print(x, end=" ")
...
0 1 2
```

Par défaut, Python ajoute un retour à la ligne à la fin d'un `print` : `end="\n"`

- Écrire un programme Python `transforme_liste(liste, n, p)` qui transforme une telle liste de taille $n \times p$ en un tableau à deux dimensions de n lignes et p colonnes :

```
>>> transforme_liste(naissances, 3, 5)
[[1, 3, 2, 3, 2],
 [2, 4, 2, 6, 4],
 [1, 3, 2, 1, 4]]
```

Listes par compréhension

5. Lire des listes par compréhension

Décrire les listes suivantes :

- `carres = [x ** 2 for x in range(5)]`
- `modulo_2 = [x for x in range(10) if x % 3 == 2]`
- On considère les mots :

```
mots = ["bonjour", "manipuler", "avant", "mercredi", "parcours"]
```

```
mot_avec_a = [mot for mot in mots if 'a' in mot]
longueurs = [len(mot) for mot in mots]
```

6. Construire une liste par compréhension

- Construire par compréhension la liste des cubes des entiers entre 3 et 10.
- Les joueurs de l'équipe sont enregistrés dans un tableau :

```
equipe = [('Marcel', 'attaquant'), ('Pierre', 'défenseur'), ('Joseph', 'Milieu'),
          ('Guy', 'attaquant'), ('Raoul', 'défenseur'), ('Gérard', 'Milieu')]
```

Construire par compréhension la liste des attaquants de l'équipe.

- Dans la variable `nombres` on a enregistré une liste d'entiers naturels.

Construire par compréhension la liste des entiers multiples de 3 parmi `nombres`.

7. Structure imbriquée et compréhension

On peut construire des structures par compréhension :

```
>>> [s * i for s in ["a", "b", "c"] for i in range(4)]
['', 'a', 'aa', 'aaa', '', 'b', 'bb', 'bbb', '', 'c', 'cc', 'ccc']
```

Remarquons l'ordre des boucles :

- La première boucle écrite est `for s in ["a", "b", "c"]`

2. La seconde boucle écrite est `for i in range(4)`

Dans la sortie :

1. la lettre “a” apparaît en premier, ensuite “b”, etc.
2. Les tailles respectives des chaînes sont 0, 1, 2, 3, 4, 0, 1, 2, 3, 4 etc.

Un autre exemple :

```
>>> [s + t for s in ["a", "b", "c"] for t in ["x", "y", "z"]]
['ax', 'ay', 'az', 'bx', 'by', 'bz', 'cx', 'cy', 'cz']
```

Questions :

1. Évaluez l'instruction suivante :

```
>>> [s * i for s in ["a", "b", "c"] for i in range(4)]
['', 'a', 'aa', 'aaa', '', 'b', 'bb', 'bbb', '', 'c', 'cc', 'ccc']
```

2. Évaluez l'instruction suivante :

```
[(i + j) for i in range(2) for j in range(2, 5)]
```

3. Construire par compréhension la liste suivante :

```
['ya', 'yb', 'yc', 'yya', 'yyb', 'yyc', 'yyya', 'yyyb', 'yyyc']
```

On peut aussi créer des structures imbriquées par ce procédé :

```
>>> [[i + j for i in range(3)] for j in range(4, 7)]
[[4, 5, 6], [5, 6, 7], [6, 7, 8]]
```

Questions

4. Évaluer l'instruction suivante :

```
[[i * j for i in range(3)] for j in range(4, 7)]
```

5. Construire par compréhension la liste suivante :

```
[[5, 6, 7, 8, 9], [6, 7, 8, 9, 10]]
```

Itérer sur un Dictionnaire

7. Les développeurs

On considère le statut des membres d'une équipe de développeurs :

```
developpeurs = {
    'Marcel': 'technicien',
    'Fanny': 'ingénieur',
    'Paul': 'ingénieur',
    'Frank': 'technicien',
    ...}
```

La fonction `alerter(nom, message)` envoie un message à un individu donné par son nom. Ses paramètres d'entrée sont des chaînes de caractères.

1. Écrire une boucle qui parcourt le dictionnaire et alerte tous les ingénieurs
2. Écrire une boucle qui parcourt le dictionnaire et compte les ingénieurs.
3. Comment récupérer la liste des prénoms de l'équipe de développeurs ?

8. Les monstres du jeu

On a enregistré dans un dictionnaires les caractéristiques des monstres qui figurent dans un jeu :

```
monstres = {
    'Grogneur': ['Tempête', 12, 14],
    'Frappé': ['Tempête', 8, 16],
    'Brûleur': ['Feu', 20, 6],
```

```
'Givré':    ['Glace', 17, 13],  
'Charbon': ['Feu', 10, 18]  
...}
```

Le premier élément de chaque liste est le type du monstre, ensuite sont indiqués sa force et sa vie.

1. Créer, à l'aide d'une boucle `for` Python, la liste des noms des monstres.
2. Proposer une boucle Python qui affiche successivement tous les types des monstres :

```
Tempête  
Tempête  
Feu  
Glace  
Feu  
...
```

3. Comment déterminer le monstre qui a le plus de vie ? Le moins de force ?

Englober dans une fonction :

```
>>> le_moins_fort(monstres)  
Frappé  
>>> le_plus_de_vie(monstres)  
Charbon
```