

5

Tuple et Tableau

Extrait du programme

THÈME : TYPES CONSTRUITS

Contenus : p -uplets.

Capacités attendus : Écrire une fonction renvoyant un p -uplet de valeurs.

Commentaires :



THÈME : TYPES CONSTRUITS

Contenus : Tableau indexé,
tableau donné en compréhension

Capacités attendus : Lire et modifier les éléments d'un tableau grâce à leurs index.

Construire un tableau par compréhension.

Utiliser des tableaux de tableaux pour représenter des matrices : notation `a[i][j]`.

Itérer sur les éléments d'un tableau.

Commentaires : Seuls les tableaux dont les éléments sont du même type sont présentés.

Aucune connaissance des tranches (slices) n'est exigible.

L'aspect dynamique des tableaux de Python n'est pas évoqué.

Python identifie listes et tableaux.

Il n'est pas fait référence aux tableaux de la bibliothèque NumPy.

Un tableau de Melinda
Copper

I. Tuple

a) Définition

D'après Wikipédia : *En mathématiques, si n est un entier naturel, alors un n -uplet ou n -uple est une collection ordonnée de n objets.*

Par exemple les coordonnées d'un point (5;1) est 2-uplet, une date de naissance (31;décembre;2000) est un 3-uplet.

En informatique, on utilise le terme anglais **tuple**. D'après Wikipédia¹, il s'agirait de l'abréviation de quint**tuple**, sext**tuple**.

D'après Wiktionnaire², ce serait le néologisme **table uplet**, une table de n -uplet.

Peut-être ne le saurons-nous jamais ...

Voici une définition plus précise de la notion de tuple dans un cadre informatique :

DÉFINITION :

Un **tuple** est un objet composé de différents éléments ordonnés, avec un indice, **non modifiables**.

b) Création d'un tuple

- Pour créer un tuple il suffit d'écrire des valeurs séparées par des virgules. Ces valeurs ne sont pas obligatoirement du même type.

```
1 >>> adaTuple = 'Ada', 'Lovelace', 10, 12, 1815
2 >>> adaTuple
3 ('Ada', 'Lovelace', 10, 12, 1815)
```

- On peut aussi mettre les valeurs entre parenthèses (D'une manière générale, on préférera cette notation).

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> adaTuple
3 ('Ada', 'Lovelace', 10, 12, 1815)
```

- Un tuple peut contenir un autre tuple...

```
1 >>> alanTuple = 'Alan', 'Turing', (23, 6, 1912)
2 >>> alanTuple
3 ('Alan', 'Turing', (23, 6, 1912))
```

- Si le tuple est composé d'une seule valeur il faut l'écrire `t=3`, ou `t=(3,)`. Attention la virgule est obligatoire. (A nouveau on privilégiera la notation avec les parenthèses)

```
1 >>> t=3,
2 >>> t
3 (3,)
```

- Pour tuple vide `t=()`.

```
1 >>> t=()
2 >>> t
3 ()
```

EN RÉSUMÉ :

Les virgules sont toujours obligatoires, pas les parenthèses... sauf pour le tuple vide.

1. <https://fr.wikipedia.org/wiki/N-uplet>

2. <https://fr.wiktionary.org/wiki/tuple>

c) Accès aux éléments

Si un tuple est non vide, ses n éléments sont indexés par les entiers $0, 1, 2, \dots, n - 1$.

- On accède à chacun de ses éléments en utilisant des crochets et l'indice en élément.

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> adaTuple[0]
3 'Ada'
4 >>> adaTuple[2]
5 10
6 >>> alanTuple = ('Alan', 'Turing', (23, 6, 1912))
7 >>> alanTuple[2]
8 (23, 6, 1912)
9 >>> alanTuple[2][0]
10 23
```

- Un dépassement de l'indice $n - 1$ renvoie une erreur.

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> adaTuple[5]
3 Traceback (most recent call last):
4   File "<console>", line 1, in <module>
5 IndexError: tuple index out of range
```

- La longueur d'un tuple est donnée par la fonction `len()`.

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> len(adaTuple)
3 5
4 >>> alanTuple = ('Alan', 'Turing', (23, 6, 1912))
5 >>> len(alanTuple)
6 3
```

- On peut accéder aux éléments en partant de la fin du tuple, l'indice -1 étant une abréviation de `len(t)-1`.

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> adaTuple[-1]
3 1815
4 >>> adaTuple[-3]
5 10
```

- Les éléments d'un tuple ne sont pas modifiable.

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> adaTuple[1] = 'comtesse de Lovelace'
3 Traceback (most recent call last):
4   File "<console>", line 1, in <module>
5 TypeError: 'tuple' object does not support item assignment
```

d) Affectation multiple

- On peut affecter à plusieurs variables en même temps à l'aide d'un tuple.

```
1 >>> a,b,c = 4,5,6
2 >>> a
3 4
```

- On peut alors échanger simplement³ les valeurs de plusieurs variables.

```
1 >>> a,b = 4,5
2 >>> a,b = b,a
3 >>> a
4 5
5 >>> b
6 4
```

- On peut enfin affecter à plusieurs variables les éléments d'un tuple. Il faut juste s'assurer d'avoir autant de variables que d'éléments dans le tuple.

```
1 >>> adaTuple = ('Ada','Lovelace',10,12,1815)
2 >>> prenom,nom,jour,mois,annee = adaTuple
3 >>> nom
4 'Lovelace'
5 >>> jour
6 10
```

e) Opérations

- Le symbole + effectue une concaténation de deux tuples (comme avec les chaînes de caractères).

```
1 >>> adaNom = ('Ada','Lovelace')
2 >>> adaNaissance = (10,12,1815)
3 >>> adaTuple = adaNom + adaNaissance
4 >>> adaTuple
5 ('Ada', 'Lovelace', 10, 12, 1815)
```

- Le symbole * répète plusieurs fois la concaténation du même tuple.

```
1 >>> tuple = ('je','me','répète')
2 >>> 3*tuple
3 ('je', 'me', 'répète', 'je', 'me', 'répète', 'je', 'me', 'répète')
```

3. Ce n'est pas forcément le cas dans d'autres langages. On a déjà vu par exemple qu'en Java on doit utiliser une variable temporaire pour échanger les valeurs de deux variables.

f) Fonction renvoyant un tuple

Division euclidienne

Nous avons déjà vu au chapitre 4, **Représentation des entiers**, que nous avons besoin de la division euclidienne, c'est-à-dire de la valeur du quotient entier ET du reste.

Nous pouvons créer une fonction qui va nous renvoyer ces 2 valeurs sous la forme d'un tuple.

Ligne 4, les parenthèses ne sont pas obligatoires mais il s'agit du tuple (`quotient`, `reste`).

On obtient les résultats suivants :

```
1 def divEuclidienne(a,b):
2     quotient = a//b
3     reste = a%b
4     return quotient, reste
```

```
1 >>> divEuclidienne(10,2)
2 (5, 0)
3 >>> divEuclidienne(13,2)
4 (6, 1)
5 >>> quotient, reste = divEuclidienne
    (13,2)
6 >>> reste
7 1
```

Une série de notes

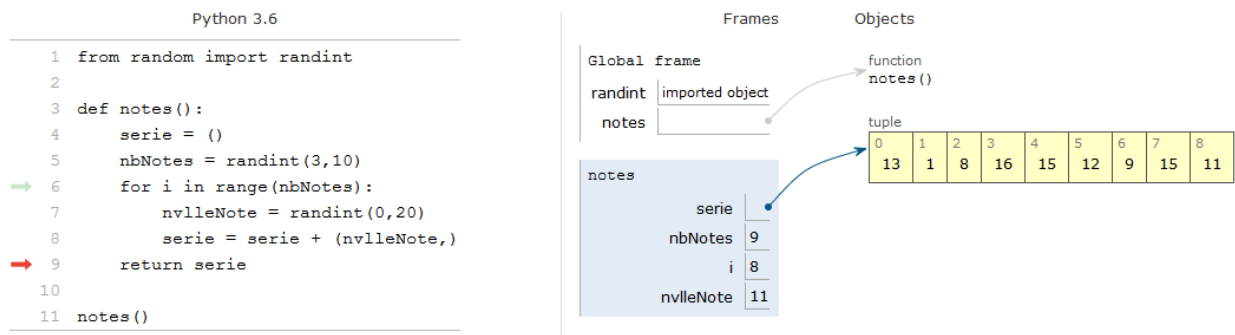
La fonction suivante permet de créer un tuple de longueur aléatoire entre 3 et 10, contenant des nombres entiers choisis aux hasards entre 0 et 20. Cette fonction va nous servir à simuler l'ensemble des notes sur un trimestre dans une matière.

```
1 from random import randint
2
3 def notes():
4     serie = ()
5     nbNotes = randint(3,10)
6     for i in range(nbNotes):
7         nvlleNote = randint(0,20)
8         serie = serie + (nvllNote,)
9     return serie
```

Nous obtenons par exemple le résultat suivant :

```
1 >>> notes()
2 (13, 1, 8, 16, 15, 12, 9, 15, 11)
```

Nous pouvons regarder ce qui se passe, pas à pas, en ligne sur le site <http://pythontutor.com/> :



- Mais Monsieur, votre truc, là, il arrête pas de modifier le tuple ! Sa longueur est de plus en plus grande !
- Oui tu as tout à fait raison, petit scarabée.
- Vous nous auriez menti ! ? ? ?
- Pas tout à fait.
- C'est-à-dire ?
- A chaque itération on crée un nouveau tuple qui va contenir la concaténation de l'ancien tuple plus le tuple contenant la nouvelle note, puis on efface de la mémoire l'ancien qui porte le même nom.
- Vous voulez dire qu'à chaque étape, le tuple *serie* n'est plus vraiment le même, bien qu'il porte le même nom ?
- Oui, petit scarabée. Tu as compris.
- Hé hé hé !
- On peut d'ailleurs s'en rendre compte si on demande l'adresse mémoire du tuple *serie* avant et après ajout de la nouvelle note. Dans le code ci-dessous, c'est aux lignes 10 et 12.

```

1 from random import randint
2
3 def notes():
4     serie = ()
5     nbNotes = randint(3,10)
6     for i in range(nbNotes):
7         nvllNote = randint(0,20)
8         print("avant",id(serie))
9         serie = serie + (nvllNote,)
10        print("après",id(serie))
11    return serie

```

La console donne :

```

1 >>> notes()
2 avant 895051038792
3 après 895125396728
4 avant 895125396728
5 après 895195801544
6 avant 895195801544
7 après 895125264496
8 (10, 7, 5)

```

- On constate bien que l'adresse mémoire du tuple *serie* n'est pas la même avant et après modification. Ce n'est donc pas le même tuple.
- Hé ! Hé ! Hé ! J'avais bon !
- Oui et maintenant c'est à toi de coder petit scarabée.
- Ups !

g) Quelques exercices sur les tuples

Exercice 1

1. Créer une fonction qui prend en argument un tuple de notes et qui renvoie un tuple donnant, dans cet ordre la plus petite note, la moyenne et la plus grande note.
 ENTRÉE : Un tuple de n notes.
 SORTIE : Un tuple donnant dans l'ordre la plus petite note, la moyenne des notes et la plus grande note.
 EXEMPLES :
`statNotes((13, 1, 8, 16, 15, 12, 9, 15, 11))` doit renvoyer `(1, 11.11, 16)`.
`statNotes((4, 2, 3, 0, 6, 15, 11, 16, 6, 13))` doit renvoyer `(0, 7.6, 16)`.
 AIDE : On peut commencer par faire une fonction qui calcule la moyenne, puis une autre qui recherche la plus petite note, enfin la note la plus grande.
2. Tester votre fonction avec la fonction `notes()` donnée précédemment.
3. Créer un programme qui crée 3 listes de notes en NSI, en Français et en Histoire-Géographie.
 En utilisant les coefficients suivants, 10 pour la NSI, 5 pour le Français et 3 en Histoire-Géographie, votre programme doit vous dire si vous aurez votre BAC ... ou pas.

Exercice 2 (Triplet Pythagoricien)

Un professeur de mathématiques fainéant cherche des triplets Pythagoriciens⁴ différents du classique (3, 4, 5). Il demande donc à ses élèves de NSI de lui écrire un programme qui en trouve aléatoirement un pour des longueurs entières comprises entre 1 et 100. Seule demande particulière, la dernière longueur doit être la plus grande (et donc être la longueur de l'hypoténuse).

- *Mais Monsieur, je le connais ce prof!*

- *Je ne pense pas petit scarabée.*

Exercice 3 (Triplet Pythagoricien **)

Et si on trouvait le nombre⁵ de triplets Pythagoriciens composés de nombres entiers inférieurs à 100 ???

Exercice 4 (Triplet Pythagoricien *)**

Et si on estimait le nombre d'essais nécessaire à notre premier programme pour trouver un triplet Pythagoricien composés de nombres entiers inférieurs à 100 ???

h) Limites

- *Mais Monsieur! on ne peut pas modifier la longueur d'un tuple ni les différentes valeurs qui la composent. C'est un peu limité votre truc!*

- *Oui, petit scarabée. Un tuple est une collection de taille constante de valeurs constantes.*

- *Ben c'est ce que je dis!*

- *C'est quand même bien pratique de pouvoir stocker un ensemble de valeurs, même si nous ne pouvons pas les modifier. D'ailleurs Ada Lovelace n'a aucune raison de changer de nom ou de date de naissance :*

`adaTuple = ('Ada', 'Lovelace', 10, 12, 1815).`

- ...

- *Nous verrons dans la suite du cours la notion de tableau qui permet de dépasser cette limite.*

- *Je suis pas pressé de voir ça... Mais Monsieur, j'y pense, votre truc est nul, parce que mon cousin il vit aux USA, et là-bas, ils inversent le jour et le mois. Avec votre tuple `adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)`, en France, on pense qu'Ada est née le 10 décembre 1815, alors qu'aux States, on pense qu'elle est née le 12 octobre 1815... Hé hé hé! (là je lui ai bien détruit son cours!!!) Hé hé hé!*

- *C'est bien, tu penses.*

- ...

- *Et, oui, avec un tuple, l'ordre dans lequel les valeurs sont données a son importance. Il faut donc se donner des règles bien strictes sur la définition de notre tuple. Par exemple, pour le point de coordonnées (2, 5), on sait que l'abscisse est 2 et l'ordonnée est 5. C'est une règle arbitraire que nous avons choisie. Mais dans d'autres cas, c'est moins clair comme tu viens de le remarquer petit scarabée.*

- *Hé hé!*

- *Nous verrons plus tard la notion de dictionnaire qui permet d'associer une clef à chaque valeurs.*

- *Oh non! Pitié!*

4. Un triplet Pythagoricien est un triplet de trois nombres correspondant aux longueurs des 3 côtés d'un triangle rectangle. Par exemple (3, 4, 5) en est car $3^2 + 4^2 = 5^2$.

5. Il y a exactement 16 triplets primitifs : (3, 4, 5) (20, 21, 29) (11, 60, 61) (13, 84, 85) (5, 12, 13) (12, 35, 37) (16, 63, 65) (36, 77, 85) (8, 15, 17) (9, 40, 41) (33, 56, 65) (39, 80, 89) (7, 24, 25) (28, 45, 53) (48, 55, 73) (65, 72, 97) d'après mon ami Wikipédia.

https://fr.wikipedia.org/wiki/Triplet_pythagoricien.

Les autres triplets sont obtenus par proportionnalité. Ainsi en multipliant par 2 (3, 4, 5), on a (6, 8, 10), en multipliant par 3, on a (9, 12, 15), ...

II. Hors programme mais parfois bien pratique : le slicing

Le **slicing** est une technique particulière qui permet de récupérer une série d'éléments d'un tuple à partir d'une borne, jusqu'à une borne, ou entre deux bornes.

- Donner tous éléments à partir de celui indexé 2 (**inclus**) : `tuple[2:]`

```
1 >>> adaTuple = ('Ada', 'Lovelace', 10, 12, 1815)
2 >>> adaTuple[2:]
3 >>> (10, 12, 1815)
```

- Donner tous éléments jusqu'à celui indexé 2 (**exclu**) : `tuple[:2]`

```
1 >>> adaTuple[:2]
2 >>> ('Ada', 'Lovelace')
```

- Donner tous éléments entre celui d'index 1 (**inclus**) et celui d'index 3 (**exclu**) : `tuple[1:3]`

```
1 >>> adaTuple[1:3]
2 >>> ('Lovelace', 10)
```

Comme vous pouvez le constater dans ces 3 exemples, la borne inférieure est toujours incluse, la borne supérieure est toujours exclue.

III. Tableau

a) Définition

Mon ami wikipédia⁶ me donne la définition suivante pour un tableau :

« En informatique, un **tableau** est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou indice, dans la séquence. C'est un type de conteneur que l'on retrouve dans un grand nombre de langages de programmation. »

...

Malheureusement cette structure n'est pas implémenté en Python.

Cool. Le chapitre est terminé.

Au revoir et à la semaine prochaine.

6. [https://fr.wikipedia.org/wiki/Tableau_\(structure_de_données\)](https://fr.wikipedia.org/wiki/Tableau_(structure_de_données))

III. Tableau

a) Définition

Bon, sérieusement, un tableau c'est quoi ?

DÉFINITION :

Un **tableau** est un ensemble ordonné, de taille fixe, d'éléments d'un même type.

On peut accéder à chaque élément d'un tableau, récupérer sa valeur et/ou la modifier.

Avec Python, nous utiliserons le type `list` qui est beaucoup plus permissif. Voyons plutôt.

b) Ensemble ordonné d'éléments de même type

En Python, une liste est un ensemble ordonné d'éléments *qui ne sont pas nécessairement d'un même type*. Ces éléments sont écrits entre crochets séparés par des virgules.

- Ce que l'on peut faire en Python, **mais** qu'il ne faut pas faire :

```
1 >>> T = [12, "azerty", 3.1415927, ("Alan", "Turing")]
```

T est une *liste* ordonnée de éléments :

.....

Bien ce soit possible en Python, **cette écriture est déconseillée, hors programme et interdite au lycée**⁷.

- Ce qu'il faut faire :

```
1 >>> T = [3, 1, 2020]
2 >>> U = ["Alan", "Turing"]
3 >>> V = [3.1415927, 1.414]
```

Les listes T, U et V contiennent des éléments ordonnés du même type. On considérera alors que ce sont des **tableaux**.

T est un tableau de

U est un tableau de

V est un tableau de

7. J'espère avoir été assez clair !

c) Les éléments sont modifiables

- On peut avoir accès à tout élément d'un tableau par son index. On peut alors récupérer cet élément et/ou le modifier. Attention ! Vous en avez maintenant l'habitude, on commence à compter à partir de 0 et si le tableau est de longueur n , le dernier élément est associé à l'index $n - 1$.

De plus, on peut compter à rebours, de la droite vers la gauche, en partant du dernier élément indexé -1.

```

1 >>> T = [3, 4, 8, 9, 10, 11, 45, 1, 6]
2 >>> T[0]
3 3
4 >>> T[1]
5 4
6 >>> T[9]
7 Traceback (most recent call last):
8   File "<console>", line 1, in <module>
9   IndexError: list index out of range
10 >>> T[8]
11 6
12 >>> T[-1]
13 6
14 >>> T[-9]
15 3

```

- En ayant un accès à un élément d'un tableau on peut aussi le modifier.

```

1 >>> T = [3, 4, 8, 9, 10, 11, 45, 1, 6]
2 >>> T[0]=2020
3 >>> T
4 [2020, 4, 8, 9, 10, 11, 45, 1, 6]

```

d) Un tableau est de taille fixe

Attention, en Python, une liste **n'est pas** de taille fixe.

- HORS - PROGRAMME** : En Python, on peut utiliser la méthode `append()` pour rajouter un élément à la fin de la liste et la méthode `pop()` pour enlever le dernier élément de la liste. Ces deux notions sont totalement hors-programme, mais parfois bien pratiques. Il est donc possible que parfois je les utilise en première. Bien évidemment, aucune connaissance sur ces 2 méthodes n'est exigibles :

```

1 >>> T = [3, 4, 8, 9, 10, 11, 45, 1, 6]
2 >>> T.append(2)
3 >>> T
4 [3, 4, 8, 9, 10, 11, 45, 1, 6, 2]
5 >>> T.pop()
6 2
7 >>> T
8 [3, 4, 8, 9, 10, 11, 45, 1, 6]

```

e) Quelques outils et méthodes intéressantes en Python

- On peut connaître la longueur d'une liste avec la fonction `len()` :

```
1 >>> T = [1,2,3]
2 >>> len(T)
3 3
```

- On peut concaténer plusieurs listes avec l'opérateur `+` :

```
1 >>> T = [1, 2, 3]
2 >>> U = [4, 5, 6]
3 >>> V = T + U
4 >>> V
5 [1, 2, 3, 4, 5, 6]
```

- On peut répéter plusieurs fois une liste avec l'opérateur `*` :

```
1 >>> T = [1, 2, 3]
2 >>> U = 3 * T
3 >>> U
4 [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- On peut inverser l'ordre d'une liste avec la méthode `reverse()` :

```
1 >>> T = [1, 2, 10, 5, 8, 3]
2 >>> T.reverse()
3 >>> T
4 [3, 8, 5, 10, 2, 1]
```

- On peut trier une liste avec la méthode `sort()` :

```
1 >>> T = [1, 2, 10, 5, 8, 3]
2 >>> T.sort()
3 >>> T
4 [1, 2, 3, 5, 8, 10]
```

IV. Construction et parcours d'un tableau

Dans cette section nous allons créer, de 2 façons différentes, un tableau contenant les 100 premiers carrés parfaits, de 0^2 à 99^2 .

a) Construction à l'aide d'un boucle pour

On commence par créer un tableau aux bonnes dimensions avec des cases vides simulées par le mot-clef `None`. On remplit ensuite le tableau avec les bonnes valeurs.

```
1 t = [None]*100
2 # on remplit le tableau
3 for i in range(100):
4     t[i] = i**2
```

b) Construction par compréhension

Cette technique permet d'écrire la boucle pour à l'intérieur des crochets de la liste. C'est plus court à écrire mais parfois plus compliqué à comprendre.

Cette définition prend la forme suivante :

```
[fonction(item) for item in iterable if condition(item)]
```

Soit, dans notre cas :

```
1 t = [i**2 for i in range(100)]
```

Et si par exemple, on veut tous les carrés parfaits pairs (donc le reste de la division par 2 est 0) :

```
1 t = [i**2 for i in range(100) if (i**2)%2==0]
```

c) Parcourir un tableau par son indice

```
1 for i in range(len(t)):
2     print(t[i])
```

d) Parcourir un tableau par ses éléments

```
1 for element in t:
2     print(element)
```

V. Algorithmes à connaître par cœur

a) Recherche du minimum dans un tableau

On suppose que le tableau contient des éléments comparables.

```
1 def mini(tab) :  
2     candidat = tab[0]  
3     for i in range(1, len(tab)) :  
4         if tab[i] < candidat :  
5             candidat = tab[i]  
6     return candidat
```

Modifier le programme suivant pour qu'il renvoie la position du plus grand élément du tableau (dans cet exercice, on considère que ce maximum est unique).

```
1 def maxi(tab) :  
2     position = .....  
3     for i in range(1, len(tab)) :  
4         .....  
5         .....  
6     return position
```

b) Calcul de la somme de tous les éléments d'un tableau

On suppose que le tableau ne contient que des nombres.

```
1 def somme(tab) :  
2     calcul = 0  
3     for i in range(len(tab)):  
4         calcul = calcul + tab[i]  
5     return calcul
```

Modifier le programme suivant pour qu'il renvoie la moyenne des éléments d'un tableau.

```
1 def moyenne(tab) :  
2     calcul = 0  
3     for i in range(len(tab)):  
4         .....  
5         .....  
6     return calcul
```

VI. Copie (HORS-PROGRAMME)

Attention, ceci est une partie très délicate du cours qui peut aboutir à de nombreuses erreurs...

Le type `list` est un objet qui permet de modifier les valeurs de chaque éléments de la liste. Malheureusement cette souplesse à un impact sur la copie de listes. Voyons plutôt :

```
1 >>> liste1 = [1, 2, 3]
2 >>> liste2 = liste1
3 >>> liste2 # tout va bien, liste2 contient bien les même éléments que liste1
4 [1, 2, 3]
5 >>> liste1[1] = 5
6 >>> liste1 # normal, on vient de modifier la liste1
7 [1, 5, 3]
8 >>> liste2 # ben ça alors, on a aussi modifier la liste2
9 [1, 5, 3]
```

En réalité, les deux noms `liste1` et `liste2` font référence à un seul et même objet. On peut le vérifier en utilisant la fonction `id()` qui nous donne l'identité de chaque objet, autrement dit l'adresse mémoire de l'objet :

```
1 >>> id(liste1)
2 676146267336
3 >>> id(liste2)
4 676146267336
```

On constate bien que les 2 noms font référence à la même adresse. C'est donc qu'il s'agit du même objet. Si on modifie `liste1`, on modifie bien `liste2`.

On peut aussi visualiser ce référencement à l'aide du site PythonTutor : <http://pythontutor.com/>



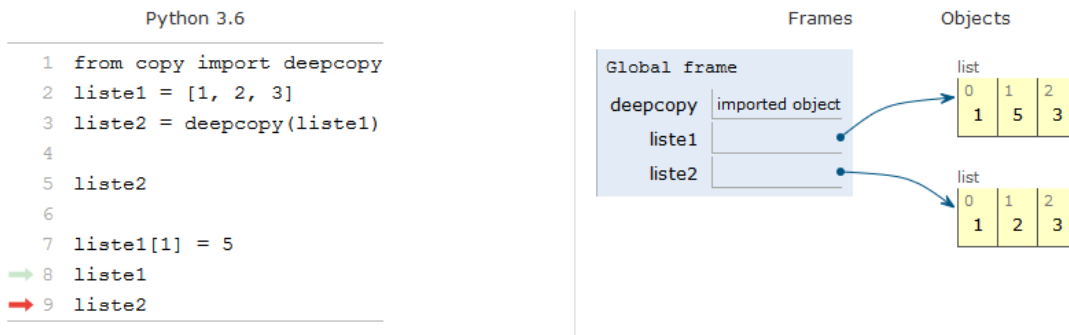
Pour obtenir deux listes différentes, il faut faire ce qu'on appelle une **copie profonde** à l'aide de la fonction `deepcopy()` de la bibliothèque `copy` :

```

1 >>> from copy import deepcopy
2 >>> liste1 = [1, 2, 3]
3 >>> liste2 = deepcopy(liste1)
4 >>> liste2 # tout va bien
5 [1, 2, 3]
6 >>> liste1[1] = 5
7 >>> liste1
8 [1, 5, 3]
9 >>> liste2 # je suis rassuré !
10 [1, 2, 3]
11 >>> id(liste1)
12 676146263304
13 >>> id(liste2) # oui ! ce n'est pas la même adresse
14 676132758536

```

A nouveau un lien vers Pythontutor pour voir ce qui se passe : <http://pythontutor.com/>



VII. Exercices sur les tableaux

Exercice 5 (Multiples, Diviseurs *)

1. Écrire une fonction `multiples1(n)` qui prend en paramètre un entier naturel non nul `n` et renvoie un tableau contenant les 10 premiers multiples non nuls de `n`.
2. Écrire une fonction `multiples2(n)` qui prend en paramètre un entier naturel non nul `n` et renvoie un tableau contenant la liste des multiples de `n` inférieurs strictement à 1000.
3. Écrire une fonction `diviseurs(n)` qui prend en paramètre un entier naturel non nul `n` et renvoie un tableau contenant la liste des diviseurs de `n`.

Exercice 6 (Multiples, Diviseurs par compréhension **)

Reprendre l'exercice précédent réécrivant les fonctions à l'aide d'une construction par compréhension.

Exercice 7 (Multiplier un tableau *)

Écrire une fonction `produit(nombres,n)` qui prend en argument un tableau de nombres `nombres` et un entier naturel non nul `n` et renvoie un nouveau tableau obtenue en multipliant chaque élément du tableau `nombres` par `n`.

Exercice 8 (Copier un tableau **)

Écrire une fonction `copie_tableau(tab)` qui prend en argument un tableau et qui renvoie un autre tableau de même longueur, contenant les mêmes éléments et dans le même ordre.

On proposera deux solutions, dont une par compréhension.

Exercice 9 (Manipulations de tableaux ***)

1. Écrire une fonction `tableau_aleatoire()` qui renvoie un tableau de 20 valeurs prises aléatoirement entre 1 et 10.
2. Écrire une fonction `occurrence(tab,n)` qui prend en argument un tableau `tab` de valeurs prises entre 1 et 10 et l'argument `n` un nombre entre 1 et 10. Cette fonction devra compter le nombre d'occurrence du nombre `n` dans le tableau `tab`.
3. Écrire une fonction `tableau_croissant(tab)` qui prend en argument un tableau `tab` de valeurs prises entre 1 et 10 et qui renvoie le tableau contenant tous les éléments du tableau `tab` sans aucune répétition, classé dans l'ordre.
4. Écrire une fonction `jointure(tab1, tab2)` qui prend en arguments deux tableaux `tab1` et `tab2` de valeurs prises entre 1 et 10. Cette fonction retourne un tableau contenant tous les éléments de `tab1` et `tab2` sans aucune répétition, classé dans l'ordre.

Exercice 10 (Manipulations de tableaux 2 ***)

1. Écrire une fonction `prefixe(tab1, tab2)` qui renvoie `True` si le tableau `tab1` est un préfixe du tableau `tab2`, autrement dit `tab2` commence exactement par les éléments de `tab1` dans le même ordre.
2. Écrire une fonction `suffixe(tab1, tab2)` qui renvoie `True` si le tableau `tab1` est un suffixe du tableau `tab2`, autrement dit `tab2` termine exactement par les éléments de `tab1` dans le même ordre.
3. Écrire une fonction `hamming(tab1, tab2)` qui renvoie le nombre d'indices auxquels les deux tableaux `tab1` et `tab2` sont différents.

VIII. Table



Une table est un tableau de tableaux.

Pour être plus précis, on peut dire qu'une table contient plusieurs lignes et que chaque ligne est un tableau.

Donc, une table est ce qu'on appelle plus communément ... un tableau à 2 dimensions !

En mathématiques, on appelle cela une matrice.

Voici donc une table :

1	2	3	4
5	6	7	8
9	10	11	12

Cette table est composée de 3 tableaux :

1	2	3	4
---	---	---	---

,

5	6	7	8
---	---	---	---

 et

9	10	11	12
---	----	----	----

.

On va pouvoir la coder en Python ainsi :

```
1 >>> table = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

La commande `table[i]` nous permet alors d'avoir accès à la *i*-ème ligne.

```
1 >>> table[1]
2 [5, 6, 7, 8]
```

La commande `table[i][j]` nous permet d'avoir accès à la *j*-ème case de la *i*-ème ligne.

```
1 >>> table[1][2]
2 7
```

IX. Exercices sur les tables

Exercice 11 (A faire à la main *)

```
1 >>> table = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

1. Que retourne la commande `table[0][1]` ?
2. Quelle commande retourne 8 ?
3. Quelle commande retourne 1 ?
4. Quelle commande retourne 4 ?

Exercice 12 (Dimension d'un table *)

Écrire une fonction `dim()` qui prend en argument une table et qui retourne sa dimension sous la forme d'un tuple.

Par exemple `dim([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])` retourne le tuple `(3,4)`, pour 3 lignes et 4 colonnes.

Exercice 13 (Carré magique ***)

Un carré magique d'ordre n est une table composé de n lignes et de n colonnes, chaque case contenant un nombre strictement positif.

Les nombres sont disposés de sorte que :

- la somme des éléments d'une ligne est toujours égale à la constante magique S ;
- la somme des éléments d'une colonne est toujours égale à la constante magique S ;
- la somme des éléments d'une diagonale est toujours égale à la constante magique S .

De plus, un carré magique contenant $n \times n$ nombres est dit normal s'il contient tout les nombres de 1 à n^2 .

Le tableau suivant est un carré magique normal.

4	9	2
3	5	7
8	1	6

En effet :

- La somme des éléments de chaque ligne vaut 15 :
 $4 + 9 + 2 = 3 + 5 + 7 = 8 + 1 + 6 = 15$
- La somme des éléments de chaque colonne vaut 15 :
 $4 + 3 + 8 = 9 + 5 + 1 = 2 + 7 + 6 = 15$
- La somme des éléments de chacune des deux diagonales vaut 15 :
 $4 + 5 + 6 = 2 + 5 + 8 = 15$

De plus ce carré magique est normal car il contient tout les nombres de 1 à $3^2 = 9$.

1. Écrire une fonction `somme_ligne(table,i)` qui calcule la somme de tous les nombres de la ligne `i`.
2. Écrire une fonction `somme_colonne(table,j)` qui calcule la somme de tous les nombres de la colonne `j`.
3. Écrire une fonction `somme_diagonale_hgbd(table)` qui calcule la somme de tous les nombres de la diagonale du coin en haut à gauche au coin en bas à droite.
4. Écrire une fonction `somme_diagonale_bgbd(table)` qui calcule la somme de tous les nombres de la diagonale du coin en bas à gauche au coin en haut à droite.
5. Écrire un programme `carre_magique(table)` qui retourne `True` si la `table` est un carré magique.
6. Écrire un programme `normal(table)` qui retourne `True` si la `table` est un carré normal (tous les nombres de 1 à n^2 sont présents).
7. Écrire un programme `carre_magique_normal(table)` qui retourne un tuple contenant deux booléens : `True` si la `table` est un carré magique et `True` si la `table` est un carré normal.

8. Vérifier vos programmes avec les tables suivantes :

2	7	6
9	5	1
4	3	8

1	5	2
3	6	4
4	7	9

8	1	6
3	5	7
4	9	2

10	51	2
60	11	52
20	61	12

2	1
3	4

4	5	11	14
15	10	8	1
6	3	13	12
9	16	2	7

4	14	15	1
9	7	6	12
5	11	10	8
16	2	3	13

Exercice 14 (Table de 1 à n^2 ***)

Écrire un programme `creer_table(n)` qui construit une table de n lignes et n colonnes remplie des nombres de 1 à n^2 , dans l'ordre.

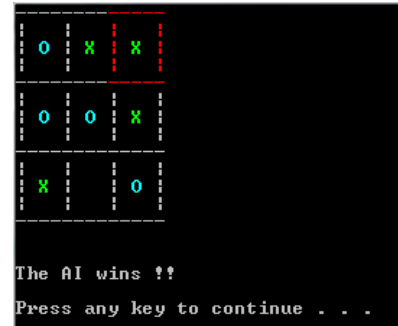
Par exemple `creer_table(2)` retourne `[[1, 2], [3, 4]]`.

De même `creer_table(3)` retourne `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.

Exercice 15 (Tic Tac Toe ****)

Programmer le jeu du tic-tac-toe.

On pourra utiliser le code ci-dessous qui permet d'afficher une grille.



```

1 grille = [['O','*',' '], [' ','O','*'], ['*','*','O']]
2
3 def affichage(table):
4     print("  A B C")
5     print("  "+"-"*7)
6     for i in range(3):
7         ligne = str(i+1)
8         for j in range(3):
9             ligne = ligne + '|' + table[i][j]
10        ligne = ligne + '|'
11        print(ligne)
12        print("  "+"-"*7)

```

Exercice 16 (Compléter un carré magique ****)

Écrire une fonction qui complète un carré magique incomplet de dimension 3×3 . Ce programme retourne un message d'alerte si le carré ne peut être magique. La table de départ devra comporter au moins une ligne une colonne déjà remplie.

Utiliser votre programme pour compléter les carrés magiques suivant :

8	1	6
	5	

		4
	5	3
		8

		2
	5	7
		6

	5	
6	1	8

	51	
60	11	52

X. Corrections des exercices

a) Exercices sur les tuples

Correction de l'exercice 1

```
1 # -*- coding: utf-8 -*-
2
3 from random import randint
4
5 def notes():
6     """
7     génère une série d'entre 3 et 10 notes sous la forme d'un tuple
8     """
9     serie = ()
10    nbNotes = randint(3,10)
11    for i in range(nbNotes):
12        nvlleNote = randint(0,20)
13        serie = serie + (nvlleNote,)
14    return serie
15
16 def moyenne(serie):
17     """
18     calcul la moyenne d'une série de notes
19     """
20    long = len(serie)
21    somme = 0
22    for i in range(long):
23        somme = somme + serie[i]
24    return round(somme/long,2)
25
26 def note_min(serie):
27     """
28     trouve le plus petit élément d'une série de notes
29     """
30    candidat = serie[0]
31    long = len(serie)
32    for i in range(1,long):
33        if candidat > serie[i]:
34            candidat = serie[i]
35    return candidat
36
37 def note_max(serie):
38     """
39     trouve le plus grand élément d'une série de notes
40     """
41    candidat = serie[0]
42    long = len(serie)
43    for i in range(1,long):
44        if candidat < serie[i]:
45            candidat = serie[i]
46    return candidat
47
48 def statNotes(serie):
49     """
50     renvoie dans cet ordre
51     la plus petite note
52     la moyenne
53     la plus grande note
54     """
55    noteMin = note_min(serie)
56    moyenneNotes = moyenne(serie)
57    noteMax = note_max(serie)
58    return noteMin, moyenneNotes, noteMax
```

```

1 # -*- coding: utf-8 -*-
2
3 from random import randint
4
5 def notes():
6     serie = ()
7     nbNotes = randint(3,10)
8     for i in range(nbNotes):
9         nvlleNote = randint(0,20)
10        serie = serie + (nvlleNote,)
11    return serie
12
13 notesNSI = notes()
14 notesFran = notes()
15 notesHG = notes()
16
17 def moyenne(serie):
18     taille = len(serie)
19     totalNotes = serie[0]
20     #on commence pour i=1 car i=0 deja fait
21     for i in range (1,taille):
22         #on rajoute une note pour le total
23         totalNotes = totalNotes + serie[i]
24     #calcul de la moyenne
25     moyenneNotes = totalNotes / taille
26     #on renvoie les resultats
27     return moyenneNotes
28
29 def bacOuPas(serieNSI,serieFran,serieHG):
30     totalAuBac = 10*moyenne(serieNSI) + 5*moyenne(serieFran) + 3*moyenne(serieHG)
31     moyenneAuBac = round(totalAuBac / (10+5+3),2)
32     if moyenneAuBac >=10 :
33         print("Bravo, avec une moyenne de",moyenneAuBac,"vous avez le BAC.")
34     else :
35         print("Ben non ! Avec une moyenne de",moyenneAuBac,"vous n'avez pas le BAC.")
36
37 #Bac ou pas
38 bacOuPas(notesNSI, notesFran, notesHG)

```

Correction de l'exercice 2 (Triplet Pythagoricien)

En rajoutant un petit compteur, on connaît le nombre d'essais effectués par l'ordinateur.

```

1 from random import randint
2
3 def tripletPytha():
4     """
5     renvoie un triplet Pythagoricien
6     les trois valeurs sont données dans l'ordre croissant
7     """
8     nTests = 0
9     #cas impossible, on rentre dans la boucle tant que
10    a, b, c = 1, 1, 1
11    while a**2 + b**2 != c**2:
12        #a est la plus petite longueur
13        a = randint(1,99)
14        #b est au moins aussi grande que a
15        b = randint(a,99)
16        #c la plus grande strictement que b

```

```

17         c = randint(b+1,100)
18         nTests = nTests+1
19         print("Je suis ton ordi. Avec ton programme, j'ai eu besoin de",nTests,"essais
pour y arriver.")
20         #on sort de la boucle, on a notre triplet
21         return a,b,c

```

Correction de l'exercice 3 (Triplet Pythagorien **)

```

1 from random import randint
2
3 def TousLesTripletsPytha():
4     """
5     le programme teste toutes les combinaisons de a, b et c
6     nombre entre 1 et 100 pour savoir quels combinaisons donnent
7     un triplet pythagoriciens.
8     Les nombres a, b et c sont données dans l'ordre croissant
9     """
10    nbTotal = 0
11    triplets = ()
12    for a in range(1,100):
13        for b in range(a,100):
14            for c in range(b+1,101):
15                if a**2 + b**2 == c**2:
16                    triplet = a,b,c
17                    nbTotal = nbTotal + 1
18                    #on rajoute le triplet au tuple triplets
19                    triplets = triplets + (triplet,)
20    print("Il y a donc",nbTotal,"de triplets Pythagoriciens composées de nombres
entiers inférieurs a 100.")
21    #on renvoie le tuple triplets contenant tous les tuples triplet
22    return triplets

```

On obtient ainsi les 52 triplets pythagoriciens rangés dans l'ordre croissant avec des nombres entiers inférieurs ou égaux à 100.

Correction de l'exercice 4 (Triplet Pythagorien ***)

```

1 from random import randint
2
3 def tripletPytha():
4     nTests =0
5     a, b, c = 1, 1, 1
6     while a**2 + b**2 != c**2:
7         a = randint(1,99)
8         b = randint(a,99)
9         c = randint(b+1,100)
10        nTests = nTests+1
11    return a,b,c,nTests
12
13 def tempsMoyen(nbRecherches):
14     nTestsTotal = 0
15     for i in range(nbRecherches):
16         nTests = tripletPytha()[3]
17         nTestsTotal = nTestsTotal + nTests
18     nTestsMoyen = round( nTestsTotal / nbRecherches , 2)
19     return nTestsMoyen

```

b) Exercices sur les tableaux

Correction de l'exercice 5 (Multiples, Diviseurs *)

```
1 def multiples1(n):
2     """
3     retourne un tableau des 10 premiers multiples non nuls de n
4     """
5     #on construit le tableau conteneur
6     T = [0]*10
7     #on le remplit
8     for i in range(1,11):
9         T[i-1] = i*n
10    return T
11
12 def multiples2_v1(n):
13     """
14     retourne un tableau contenant tous les multiples non nuls de n
15     inférieurs à 1000
16     """
17     #on calcule la taille du tableau
18     taille = 1000//n
19     #on construit le tableau conteneur
20     T = [0]*taille
21     #on le remplit
22     for i in range(1,taille+1):
23         T[i-1] = i*n
24     return T
25
26 def multiples2_v2(n):
27     """
28     retourne un tableau contenant tous les multiples non nuls de n
29     inférieurs à 1000
30     dans cette version, on utilise la méthode append() donc on
31     triche un peu car notre tableau n'est pas de taille fixe
32     c'est donc une liste
33     """
34     T = []
35     i = 1
36     while i*n < 1000:
37         T.append(i*n)
38         i = i + 1
39     return T
40
41 def diviseurs(n):
42     """
43     retourne un tableau contenant tous les diviseurs de n
44     cette fois-ci, on ne peut éviter la méthode append()
45     car on ne connaît pas apriori le nombre de diviseurs d'un nombre
46     """
47     T = []
48     diviseur = 1
49     while diviseur <= n :
50         if n%diviseur == 0 :
51             #le reste de la division est 0
52             T.append(diviseur)
53             diviseur += 1
54     return T
```


Correction de l'exercice 6 (Multiples, Diviseurs par compréhension **)

```

1 def multiples1(n):
2     T = [i*n for i in range(1,11)]
3     return T
4
5 def multiples2_v1(n):
6     T = [i*n for i in range(1,1001) if i*n<1000]
7     return T
8
9 def multiples2_v2(n):
10    taille = 1000//n
11    T = [i*n for i in range(1,taille+1)]
12    return T
13
14 def diviseurs(n):
15    return [d for d in range(1,n+1) if n%d==0]

```

Correction de l'exercice 7 (Multiplier un tableau *)

```

1 def produit(nombres,n):
2     """
3     renvoie un tableau contenant toutes les valeurs du tableau nombres
4     multipliées par n
5     """
6     taille = len(nombres)
7     T = [0]*taille
8     for i in range(taille):
9         T[i] = nombres[i]*n
10    return T
11
12 #sinon le même mais par compréhension
13 def produit_v2(nombres,n):
14     T = [element*n for element in nombres]
15     return T

```

Correction de l'exercice 8 (Copier un tableau **)

```

1 def copie_tableau(tab) :
2     taille = len(tab)
3     t = [None]*taille
4     for i in range(taille):
5         t[i] = tab[i]
6     return t
7
8 # par compréhension
9 def copie_tableau2(tab) :
10    t = [tab[i] for i in range(len(tab))]
11    return t

```

Correction de l'exercice 9 (Manipulations de tableaux ***)

```

1 from random import randint
2
3 def tableau_aleatoire():
4     """

```

```

5     génère un tableau de 20 valeurs pris aléatoirement entre 1 et 10
6     """
7     T = [randint(1,10) for i in range(20)]
8     return T
9
10    def occurrence(tab, n):
11        """
12        entrées :
13            - tab est un tableau d'entiers pris entre 1 et 10
14            - n est un entier pris entre 1 et 10
15        rôle :
16            la fonction renvoie le nombre d'occurrence de n dans tab
17        sortie :
18            un entier
19        """
20        nb = 0
21        for element in tab:
22            if element == n:
23                nb = nb + 1
24        return nb
25
26    def tableau_croissant(tab):
27        """
28        entrée :
29            tab est un tableau d'entiers pris entre 1 et 10
30        rôle :
31            la fonction renvoie un tableau contenant une et une seule fois
32            chaque éléments de tab, dans l'ordre croissant
33        sortie :
34            un tableau
35        """
36        T=[]
37        for i in range(1,11):
38            #on teste si i est présent dans tab
39            index = 0
40            present = False
41            while (index<len(tab)) and not present :
42                present = (tab[index] == i)
43                index = index + 1
44            if present :
45                #la valeur i est présente on la rajoute
46                T.append(i)
47        return T
48
49    def jointure(tab1,tab2):
50        """
51        entrées :
52            tab1 et tab2 deux tableaux d'entiers pris entre 1 et 10
53        rôle :
54            la fonction renvoie un unique tableau contenant une et une seule fois
55            chaque éléments de tab, dans l'ordre croissant
56        sortie :
57            un tableau
58        """
59        tab = tab1 + tab2
60        return tableau_croissant(tab)

```

Correction de l'exercice 10 (Manipulations de tableaux 2 ***)

Non corrigé.

c) Exercices sur les tables

Correction de l'exercice 11 (A faire à la main *)

```
1 >>> table = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

1. Que retourne la commande `table[0][1]` ? 2
2. Quelle commande retourne 8 ? `table[1][3]`
3. Quelle commande retourne 1 ? `table[0][0]`
4. Quelle commande retourne 4 ? `table[0][3]`

Correction de l'exercice 12 (Dimension d'un table *)

```
1 def dimensions(table):
2     '''
3     fonction qui retourne sous la forme d'un tuple les dimensions d'une table
4     entrée :
5         une table soit un tableau de tableaux
6     sortie :
7         un tuple
8     '''
9     nb_lignes = len(table)
10    nb_colonnes = len(table[0])
11    return (nb_lignes, nb_colonnes)
```

Correction de l'exercice 13 (Carré magique ***)

```
1 def somme_ligne(table,i):
2     somme = 0
3     for j in range(len(table)):
4         somme = somme + table[i][j]
5     return somme
6
7 def somme_colonne(table,j):
8     somme = 0
9     for i in range(len(table)):
10        somme = somme + table[i][j]
11    return somme
12
13 def somme_diagonale_hgbd(table):
14     somme = 0
15     for d in range(len(table)):
16        somme = somme + table[d][d]
17    return somme
18
19 def somme_diagonale_bgbd(table):
20     somme = 0
21     for d in range(len(table)):
22        somme = somme + table[d][-d]
23    return somme
24
25 def carre_magique(table):
26     nb_magique = somme_ligne(table,0)
27     reponse = True
28     #on teste toutes les lignes
29     i = 0
```

```

30     while i<len(table) and reponse == True:
31         reponse = (nb_magique == somme_ligne(table,i))
32         i = i + 1
33     #on teste toutes les colonnes
34     i = 0
35     while i<len(table) and reponse == True:
36         reponse = (nb_magique == somme_colonne(table,i))
37         i = i + 1
38     #on teste les deux diagonales
39     if reponse == True:
40         reponse == (nb_magique == somme_diagonale_hgbd(table))
41     if reponse == True:
42         reponse == (nb_magique == somme_diagonale_hgbd(table))
43     return reponse

```

Correction de l'exercice 14 (Table de 1 à n^2 ***)

```

1 def creer_table(n):
2     """
3     génère une table de dimension nxn
4     contenant dans l'ordre, tous les nombres de 1 à n**2
5     """
6     T = []
7     for i in range(0,n):
8         #création de i-ème ligne
9         ligne = [0]*n
10        #remplissage de la ligne
11        for j in range(0,n):
12            ligne[j] = i*n + j + 1
13        #on rajoute la ligne à la table
14        T.append(ligne)
15    return T

```

Correction de l'exercice 15 (Tic Tac Toe ****)

Non corrigé.

Correction de l'exercice 16 (Compléter un carré magique ****)

Non corrigé.