1. TP 08 Les tableaux Exercices

TD n°8: Les types
construits: tuples et
tableaux

Thème 4:
Langages et
Programmation

EXERCICES

Lien Capytale

Enoncé

On considère la variable repertoire suivante.

```
repertoire = [('Michel', '0210101010'), ('Marjorie', '0211111111'), ('Perrine', '0212121212')]
```

- 1. Quel est le type de la variable repertoire ?
- 2. Quel est le type de la variable repertoire[0] ?
- 3. Qu'affiche l'instruction repertoire[1][1]?
- 4. Quelle instruction permet d'accéder au numéro de téléphone de Michel ?
- 5. Quel est le type de la variable repertoire[2][0] ?

- La variable repertoire est de type list
- La variable repertoire[0] est de type tuple
- L'instruction repertoire[1][1] affiche 0211111111
- Pour accéder au numéro de téléphone de Michel, il faut utiliser l'instruction repertoire[0][1]
- La variable repertoire[2][0] est de type str

Répondez aux questions suivantes **puis** exécutez la cellule pour vérifier.

1. Quel est le tableau construit par les instructions suivantes ?

```
t1 = [0]*10
for i in range(10):
   t1[i] = i
```

1. Quel est le tableau construit par les instructions suivantes ?

```
t2 = [i+1 for i in range(8)]
```

1. Quel est le tableau construit par les instructions suivantes ?

```
t3 = [i*i for i in range(5, 10)]
```

1. Quel est le tableau construit par les instructions suivantes ?

```
t4 = [0]*50
for i in range(len(t4)):
    t4[i] = i % 2
```

Enoncé

Complétez la fonction suivante pour qu'elle échange les valeurs d'index i et j du tableau tab.

Exemple

```
T = [3, 8, 12, 9, 56, 14, 22, 7, 13, 41]
echange(T, 2, 4)
T
[3, 8, 56, 9, 12, 14, 22, 7, 13, 41]
```

```
def echange(tab,i,j):
    pass
    return tab

# Tests
T = [3, 8, 12, 9, 56, 14, 22, 7, 13, 41]
echange(T, 2, 4)
T
```

```
def echange(tab,i,j):
    temp=tab[j]
    tab[j]=tab[i]
    tab[i]=temp
    return tab
```

Enoncé

On considère le tableau t suivant.

```
t = [0, 2, 4, 6, 8, 10]
```

- 1. Ecrivez un programme qui parcourt *par indice* le tableau et affiche tous ses éléments.
- 1. Ecrivez un programme qui parcourt *par valeur* le tableau et affiche ses éléments.
- 1. Ecrire un programme qui calcule la somme des élèments du tableau.

Solution

1. Parcours par indice

```
for indice in range(len(t)):
    print(t[indice])
```

1. Parcours par valeur

```
for elt in t:
    print(elt)
```

2. Calcul de la somme des termes d'une suite

```
somme=0
for valeur in t:
    somme=somme+valeur # ou somme+=valeur
```

Exercice 8.5:

Enoncé

On considère le tableau défini par compréhension suivant.

```
T = [i for i in range(15)]
```

- 1. Que contient le tableau T ?
- 2. Ecrivez un programme qui parcourt le tableau T et qui affiche uniquement les multiples de 2 qu'il contient.

```
for valeur in T:
    if valeur %2==0: #on teste si valeur est pair que
montrant que le reste de la division euclidienne par 2
vaut 0
    print(valeur)
```

Enoncé

On rappelle que l'on peut construire un tableau en utilisant une boucle for (en ayant au préalable créé un tableau de la bonne taille avec une valeur arbitraire) ou en le définissant par compréhension.

Utilisez les deux méthodes pour construire les tableaux demandés.

- 1. Un tableau qui contient tous les entiers compris entre 0 et 1000.
- 2. Un tableau qui contient tous les entiers compris entre 5 et 15.
- 3. Un tableau qui contient tous les nombres pairs compris entre 0 et 50.
- 4. Un tableau qui contient les carrés de tous les entiers compris entre 1 et 100 (c'est-à-dire 1, 4, 9, 16, etc.)

```
# Q1 avec méthode 1

# Q1 avec méthode 2

# Q2 avec méthode 1

# Q2 avec méthode 2

# Q3 avec méthode 1

# Q4 avec méthode 1

# Q4 avec méthode 2
```

```
#Q1 Méthode 1
 tab1=[] #initialisation
 for k in range(0,1001):
      tab1.append(k)
 #Q1 Méthode 2 par compréhension
 tab1=[k for k in range(0,1001)]
 #Q2 Méthode 1
 tab2=[] #initialisation
 for k in range(5,16):
      tab2.append(k)
 #Q2 Méthode 2 par compréhension
 tab2=[k for k in range(5,16)]
 #Q3 Méthode 1
 tab3=[] #initialisation
 for k in range(0,51,2):
      tab3.append(k)
 #Q3 Méthode 2 par compréhension
 tab3=[k \text{ for } k \text{ in range}(0,51,2)]
ou
 #Q3 Méthode 2 par compréhension
 tab3=[k for k in range(51) if k%2==0]
 #Q4 Méthode 1
 tab4=[] #initialisation
 for k in range(1,101):
      tab4.append(k**2)
```

```
#Q4 Méthode 2 par compréhension
tab4=[k**2 for k in range(1,101)]
```

Exercice 8.7 : Moyenne annuelle

Enoncé

Compléter la fonction moyenne qui prend en argument un tableau de notes et qui renvoie la moyenne de ce tableau.

```
notes=[12,10,15,18,8]
moyenne(notes)
>>>12.6
```

```
def moyenne(notes):
    ...
    ...
```

```
def moyenne(notes):
    somme=0
    nb_notes=len(notes)
    for note in notes:
        somme+=note
    moy=somme/nb_notes
    return moy
```

Exercice 8.8 : Moyenne coefficientée

Enoncé

Voici les premières notes de l'année obtenues par un élève en NSI :

Note	coefficient
15	1
13	2
8	1
18	1

- 1. Ces notes et coefficients sont d'abord stockées dans deux tableaux comme ci-dessous.
 - a. Quelle instruction permet d'accéder la troisième note ?
 - b. Quelle instruction permet d'accéder au coefficient de la deuxième note ?

```
notes = [15, 13, 8, 18]
coeff = [1, 2, 1, 1]
```

2. Ecrire une fonction moyenne Coeff (notes, coeffs) qui prend en paramètres deux tableaux et qui renvoie la moyenne coefficientée correspondante.

```
def moyenneCoeff(notes,coeffs):
    sommenotes=0
    sommecoeffs=0
    for i in range(...):
        ...
```

3. Ces notes et coefficient sont maintenant stockées dans un tableau de tableaux.

Quelle instruction permet d'accéder à la troisième paire notecoefficient ?

Quelle instruction permet d'accéder la troisième note ? Quelle instruction permet d'accéder au coefficient de la deuxième note ?

```
notes_coeff = [[15, 1], [13, 2], [8, 1], [18, 1]]
```

4. Ecrire une fonction moyenneCoeff(tab) qui prend en paramètre un tableau et qui renvoie la moyenne coefficientée correspondante.

```
def moyenne(notes, coeffs):
    sommenotes=0
    sommecoeffs=0
    nb2notes=len(notes)
    for indice in range(nb2notes):
        sommenotes+=notes[indice]
        sommecoeffs+=coeffs[indice]
    moy=sommenotes/sommecoeffs
    return moy
```

```
def moyenneCoeff(tab):
    sommenotes=0
    sommecoeffs=0
    nb2notes=len(tab)
    for indice in range(nb2notes):
        sommenotes+=tab[indice][0]*tab[indice][1] #les
notes ont toujours pour position 0 dans le tableau
extrait et les coeffs ont pour position1
        sommecoeffs+=tab[indice][1]
    moy=sommenotes/sommecoeffs
    return moy
```

Exercice 5.9:

Enoncé

Répondez aux questions suivantes **puis** exécutez la cellule pour vérifier.

1. Quel est le tableau construit par compréhension de la façon suivante ?

```
tab = [[j**2 for j in range(5)] for i in range(3)]
tab
```

1. Quel est le tableau construit par compréhension de la façon suivante ?

```
tab = [[2*i + j \text{ for } j \text{ in } range(2)] \text{ for } i \text{ in } range(4)]tab
```

Exercice 8.10:

Enoncé

On a mémorise dans le tableau grille l'état d'un jeu de morpion. Voici le contenu du jeu après deux tours.

- 1. C'est au joueur « O » de jouer et il compte mettre un '0' au centre de la grille. Quelle instruction faut-il écrire ? Vérifiez.
- 2. Vous avez du constater qu'en affichant le tableau, celui ne s'affiche pas comme on souhaiterait (comme la grille du jeu). Pour pallier à ce souci, il suffit de parcourir le tableau ligne par ligne et afficher cette ligne. Ecrivez le programme correspondant.

3. Approfondissement

Pour ceux qui le désire, écrire un programme permettant de joueur au morpion.

Solution

```
grille[1][1]
```

```
for ligne in grille:
    print(ligne)
```

1.1 Parcours tableaux à plusieurs dimensions

Exercice 8.11:

Enoncé

Ecrire une fonction ajoutNumero(rep, nom, numero) qui prend pour paramètre un répertoire, un nomn et un numero de téléphone et qui permet d'enregistrer un nouveau nom ainsi que son numéro dans le répertoire

```
repertoire = [['Michel', '0210101010'], ['Marjorie', '0211111111'], ['Perrine', '0212121212']]
```

```
def ajoutNumero(rep, nom, numero):
    rep.append([nom, numero]]
    return rep
```

Exercice 8.12:

Enoncé

Ecrivez une fonction somme(t) qui renvoie la somme des éléments du tableau d'entiers t.

Exemple:

```
t=[1,2,3,5,7,10]
somme(t)
>>> 28
```

```
def ajoutNumero(rep, nom, numero):
    rep.append([nom, numero]]
    return rep
```

Exercice 8.13 : cadavre exquis

Enoncé

Le cadavre exquis est un « jeu qui consiste à faire composer une phrase, ou un dessin, par plusieurs personnes sans qu'aucune d'elles ne puisse tenir compte de la collaboration ou des collaborations précédentes.»

Remarque : la méthode choice du module random permet de tirer au sort un élément dans un itérable non vide (et donc dans un tableau non vide). Vous pouvez exécuter plusieurs fois la cellule de code cidessous pour tester :

```
from random import choice

tab = ['chat', 'chien', 'cheval', 'vache', 'cochon']
animal = choice(tab)
animal
```

Compléter la fonction cadavre_exquis ci-dessous qui prend en argument : - un tableau de noms non vide, - un tableau de verbes non vide, - un tableau de compléments non vide,

et qui renvoie une phrase aléatoire construite selon le principe du cadavre exquis : en piochant un nom dans le tableau de noms, puis en piochant un verbe dans le tableau de verbes et enfin en piochant un complément dans le tableau de compléments.

```
from random import choice

def cadavre_exquis(noms, verbes, complements):
    nom = ...
    verbe = ...
    complement = ...
    phrase = ...
    return ...
```

Solution

```
from random import choice

def cadavre_exquis(noms, verbes, complements):
    nom = noms
    verbe = verbes
    complement = complements
    phrase = choice(nom)+choice(verbe)+choice(complement)
    return phrase
```

Tester votre fonction en complétant l'exemple ci-dessous.

```
n = ['Le chasseur ', 'La présidente ', 'Le lapin ', 'Pikachu ', 'La
comète ', 'La licorne ']
v = ['mange ', 'cherche ', 'entretient ', 'admire ', 'vénère ',
'admoneste ']
c = ['la saucisse.', 'la forêt.', 'la casquette.', 'la bicyclette.', 'le
micro-processeur.']
cadavre_exquis(n, c,v)
```

Exercice 8.14 : Travail sur les listes en compréhension

Enoncé

Créez une liste Liste1 de 15 entiers aléatoires compris entre 0 et 20 à l'aide d'une boucle.

Aide : Nous allons utiliser le module random qui génère des nombres pseudo-alétoires.

Commencez par importer la fonction randint du module avec l'instruction :

from random import randint à placer en début de cellule.

L'instruction randint(a,b) retournera un entier alétoire compris entre a et b (bornes comprises)

Vers la documentation du module

```
#un exemple :
from random import randint
Liste1 = []

for k in range(15):
    alea=randint(0,20)
    Liste1.append(alea)
print(Liste1)
```

Créez une liste Liste2 de 30 entiers aléatoires compris entre 0 et 20 à l'aide d'une compréhension de liste.

```
Liste2 = # Votre code ici
print(Liste2)
```

Créez une fonction moyenne qui prend en paramètre une liste de nombres et qui renvoie la moyenne de cette liste.

Testez votre fonction à l'aide des assert de la cellule suivante.

```
def moyenne(liste):
    pass
return moyenne
```

```
# Méthode de TEST

assert moyenne(Liste1) == sum(Liste1)/len(Liste1)
assert moyenne(Liste2) == sum(Liste2)/len(Liste2)
```

Un méchant professeur veut baisser le notes de ses élèves de 10 % Créez une liste Liste3 obtenue en diminuant de 10 % les nombres de la liste Liste1 avec une compréhension de liste Testez votre fonction à l'aide des assert de la cellule suivante.

```
Liste3 = # Votre code ici
Liste3
```

```
for index in range(len(Liste1)):
    assert Liste3[index] == Liste1[index]*0.9
```

Le même professeur cherche à écrire des appréciation automatiques en fonction de la note.

Il souhaite voir écrit "Pas terrible" si la note est inférieure ou égale à 10 et "Pas mal" sinon.

Créez une liste Appreciations à partir de la liste Liste1 à l'aide d'une comprehension de liste

```
Aide: [<expression> if <condition> else <autre_expression>
for <element> in <iterable>]
```

```
Appreciations = # Votre code ici
```

```
print(Liste1)
print(Appreciations)
```

Le professeur souhaite maintenant créer une fonction app2 qui prend en paramètre un nombre et qui renvoie un tuple constitué de ce

nombre et d'une appréciation. L'appréciation sera : - "Bon travail" si la note est supérieure ou égale à 15; - "Fais encore des efforts" si la note est entre 10 et 15 (non compris); - "Au travail !" si la note est inférieure ou égale à 10.

Ecrivez la fonction ci-dessous

```
def app2(nombre):
    pass
    # Votre code ici
```

Créez une liste Appreciations2 à partir de la liste Liste2 contenant les notes et les appréciation à l'aide d'une compréhension de liste

```
Appreciations2 = [app2(nb) for nb in Liste2]
print(Appreciations2)
```

```
from random import randint
liste2=[randint(0,20) for k in range(30)]
```

```
def moyenne(liste):
    somme=0
    for elt in liste:
        somme+=elt
    moy=somme/len(liste)
    return moy
```

```
Appreciations=['Pas terrible' if elt<10 else 'Pas mal' for elt in Liste1 ]
```

```
def app2(nombre):
    if nombre<=10:
        return (nombre, 'Au travail !'
    elif nombre >=15:
        return (nombre, 'Bon travail')
```

```
else:
return (nombre, 'Fais encore des efforts')
```

Exercice 8.15: Travail sur les listes de listes.

Les carrés magiques : prolongement de l'exercice vu dans la leçon.

Première étape - Créer une fonction <code>Gen_carre</code> qui prend en paramètre un entier naturel non nul n, et qui renvoie un tableau carré (liste de liste) contenant les entiers de 1 à n^2 entiers "mélangés"

```
Par exemple: Gen_carre(3) porra retourner [[2,7,6], [9,5,1], [4,3,8]]
```

Vous allez utiliser le module random et en particulier la fonction shuffle qui prend en paramètre une liste et qui la mélange "sur place". Par exemple liste = [0,1,2,3] suivi de shuffle(liste) va mélanger les éléments de liste et, par exemple, on obtiendra liste = [3,0,1,2].

```
from random import shuffle, randint

def Gen_carre(n):

carre3 = Gen_carre(3)
print(carre3)
```

Deuxième étape : Ecrire une fonction $somme_ligne$ qui prend en paramètres un tableau carré d'entiers et un entier n et qui retourne la somme de la ligne d'index n

```
def somme_ligne(carre : list,n : int):

# Tests de la fonction somme_ligne

carre3 = [[2, 7, 6],
```

```
[9, 5, 1],
[4, 3, 8]]

carre4 = [[4, 5, 11, 14],
        [15, 10, 8, 1],
        [7, 3, 13, 12],
        [9, 16, 2, 6]]

assert somme_ligne(carre3,0) == 15
assert somme_ligne(carre3,1) == 15
assert somme_ligne(carre3,2) == 15
assert somme_ligne(carre4,0) == 34
assert somme_ligne(carre4,0) == 34
assert somme_ligne(carre4,3) == 33
```

Troisième étape : Ecrire une fonction verif_lignes qui prend en paramètres un tableau carré d'entiers et qui retourne un booléen indiquant si toutes les lignes ont la même somme.

```
def verif_lignes(carre : list):

# Tests de la fonction verif_lignes

assert verif_lignes(carre3) == True
assert verif_lignes(carre4) == False
```

Quatrième étape : Faire de même avec les colonnes.

Vous avez la liberté pour construire les fonctions que vous souhaitez, ajoutez des docstrings et des tests à vos fonctions.

```
def somme_colonne(carre : list,index_colonne : int):

assert somme_colonne(carre3,0) == 15
assert somme_colonne(carre3,1) == 15
assert somme_colonne(carre3,2) == 15
assert somme_colonne(carre4,0) == 35
assert somme_colonne(carre4,1) == 34

def verif_colonnes(carre : list):
```

```
assert verif_colonnes(carre3) == True
assert verif_colonnes(carre4) == False
```

Cinquième étape : Vérifier de même les deux diagonales

```
def verif_diag(carre : list):

assert verif_diag(carre3) == True
assert verif_diag(carre4) == False
```

Dernière étape : Ecrire une fonction qui prend en paramètre un tableau carré d'entiers et qui retourne un booléen indiquant si le carré est magique

```
def verif_carre(carre):
    assert verif_carre(carre3) == True
    assert verif_carre(carre4) == False
```

Bonus : à l'aide de votre fonction Gen_carre, trouvez des carrés magiques d'ordre 3, puis 4.

Conseil: soyez patient!

```
# Voici une exemple de solution (Aucune inquiétude, je ne
demandais pas de trouver ce type de programme mais
d'essayer à la main
def carre_magique_ordre3():
carre_magique_ordre3()
```