TP-Tableaux-Corrige

October 14, 2020

1 Corrigé du TP sur les tableaux

[3]:
```
# pour exécuter Python-Tutor dans les cellules
from metakernel import register_ipython_magics
register_ipython_magics()
```

Les énoncés des exercices sont suivis des corrigés.

1.1 Exercice 2 *QCM type E3C*

1. On exécute le code suivant :

```
t = [1,2,3,4,5,6,7,8,9]

v = [c for c in t if c%3 == 0]
```

Quelle est la valeur de la variable `v` à la fin de cette exécution ?

- **Réponse A :** 18 **Réponse B :** `[1,4,7]`
- **Réponse C :** `[3,6,9]` **Réponse D :** `[1,2,3,4,5,6,7,8,9]`

2. On définit : `resultat = [ i*2 for i in range(10) ]`.

Quelle est la valeur de `resultat` ?

- **Réponse A :** `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
- **Réponse B :** `[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]`
- **Réponse C :** `[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]`
- **Réponse D :** `[2, 4, 6, 8, 10, 12, 14, 16, 18]`

3. On considère la fonction suivante :

```
def somme(tab):
    s = 0
    for i in range(len(tab)):
        .....
    return s
```

Par quelle instruction faut-il remplacer les points de suspension pour que l'appel `somme([10,11,12,13,14])` renvoie 60 ?

- **Réponse A :** `s = tab[i]` **Réponse B :** `s = s + tab[i]`
- **Réponse C :** `tab[i] = tab[i] + s` **Réponse D :** `s = s + i`

1.1.1 Corrigé de la question 1

[16]:
```
%%tutor

t = [1,2,3,4,5,6,7,8,9]
v = [c for c in t if c%3 == 0]
```

```
<IPython.lib.display.IFrame at 0x7f16642e2f70>
```

1.1.2 Corrigé de la question 2

[17]:
```
%%tutor

resultat = [ i*2 for i in range(10) ]
```

```
<IPython.lib.display.IFrame at 0x7f16642e2ca0>
```

1.1.3 Corrigé de la question 3

[15]:
```
def somme(tab):
    s = 0
    for i in range(len(tab)):
        s = s + tab[i]
    return s


#Tests unitaires
from random import randint
for size in range(1, 5):
    for run in range(10):
        tab = [randint(-100, 100) for _ in range(size)]
        assert(somme(tab)) == sum(tab), "Echec sur {}".format(tab)
print("Tests unitaires réussis")
```

```
Tests unitaires réussis
```

1.2 Exercice 3

1. La fonction suivante doit calculer la moyenne d'un tableau de nombres, passé en paramètre. Avec quelles expressions faut-il remplacer les points de suspension pour que la fonction soit correcte ? *Auteur Sylvie Genre*

```
def moyenne(tableau):
    total = ...
    for valeur in tableau:
        total = total + valeur
    return total / ...
```

2. Donner la valeur des expressions Python suivantes :

```
>>> [1, 2, 3] + [4, 5, 6]
>>> 2 * [1, 2, 3]
```

1.2.1 Corrigé de la question 1

[20]:
```
def moyenne(tableau):
    total = 0
    for valeur in tableau:
        total = total + valeur
    return total / len(tableau)

#Tests unitaires
from random import randint
for size in range(1, 5):
    for run in range(10):
        tab = [randint(-100, 100) for _ in range(size)]
        assert(moyenne(tab)) == sum(tab)/len(tab), "Echec sur {}".format(tab)
print("Tests unitaires réussis")
```

```
Tests unitaires réussis
```

1.2.2 Corrigé de la question 2

[2]:
```
[1, 2, 3] + [4, 5, 6]
```

[2]:
```
[1, 2, 3, 4, 5, 6]
```

[3]:
```
2 * [1, 2, 3]
```

[3]:
```
[1, 2, 3, 1, 2, 3]
```

1.3 Exercice 4

1. Écrire une fonction Python `smul` à deux paramètres, un nombre et une liste de nombres, qui multiplie chaque élément de la liste par le nombre et renvoie une nouvelle liste :

```
>>> smul(2, [1, 2, 3])
[2, 4, 6]
```

2. Écrire une fonction Python `vsom` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes :

```
>>> vsom([1, 2, 3], [4, 5, 6])
[5, 7, 9]
```

3. Écrire une fonction Python `vdif` qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la différence terme à terme de ces deux listes (la deuxième moins la première).

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

4. Écrire une fonction Python `vprod` qui prend en paramètre deux listes de nombres de même longueur et qui retourne une nouvelle liste constituée des produits terme à terme de ces deux listes.

```
>>> vprod([1, 2, 3], [4, 5, 6])
[4, 10, 18]
```

1.3.1 Corrigé de l'exercice 4

[26]:
```python
def smul(k, tab):
    """
    Paramètres :
        k un nombre de type int ou float
        tab est un tableau de type list contenant des  nombres de type int ou␣
 ↪float
    Précondition :
        tab non vide
    Valeur renvoyée :
        un tableau de type list contenant des  nombres de type int ou float
    Postcondition :
        le tableau renvoyé est constitué des produits
        de chaque élément de tab par k
    """
    assert len(tab) != 0, "tab doit être non vide"
    return [k * e for e in tab]


def vsom(tab1, tab2):
    """
    Paramètres :
        tab1 et tab2 des tableaux de nombres de type int ou float
    Préconditions :
        tab1 non vide et len(tab1) == len(tab2)
    Valeur renvoyée :
        un tableau de type list contenant des  nombres de type int ou float
    Postcondition :
        le tableau renvoyé est constitué des sommes
        terme à terme des éléments de tab1 et tab2
    """
    assert len(tab1) > 0 and len(tab1) == len(tab2); "tab1 et tab2 doivent être␣
 ↪de même longueur"
    return [tab1[k] + tab2[k] for k in range(len(tab1))]
```

```python
def vdiff(tab1, tab2):
    """
    Paramètres :
        tab1 et tab2 sont des tableaux de type list contenant des  nombres de␣
 ↪type int ou float
    Précondition :
        tab1 non vide et len(tab1) == len(tab2)
    Valeur renvoyée :
        un tableau de type list contenant des  nombres de type int ou float
    Postcondition :
        le tableau renvoyé est constitué des différences terme à terme
        des éléments de tab2 et tab1 (la deuxième moins la première)
    """
    assert len(tab1) > 0 and  len(tab1) == len(tab2); "tab1 et tab2 doivent␣
 ↪être de même longueur"
    return [tab2[k] - tab2[k] for k in range(len(tab1))]

def vprod(tab1, tab2):
    """
    Paramètres :
        tab1 et tab2 sont des tableaux de type list contenant des  nombres de␣
 ↪type int ou float
    Précondition :
        tab1 non vide et len(tab1) == len(tab2)
    Valeur renvoyée :
        un tableau de type list contenant des  nombres de type int ou float
    Postcondition :
        le tableau renvoyé est constitué des produits
        terme à terme des éléments de tab2 et tab1
    """
    assert len(tab1) > 0 and len(tab1) == len(tab2); "tab1 et tab2 doivent être␣
 ↪de même longueur"
    return [tab2[k] * tab2[k] for k in range(len(tab1))]
```

1.4 Exercice 5

1. Écrire une fonction `produit(tab)` qui retourne le produit des éléments d'un tableau de nombres `tab`.
2. Écrire une fonction `all_positive(tab)` qui retourne un booléen indiquant si tous les éléments du tableau de nombres `tab` sont strictement positifs.
3. Écrire une fonction `any_positive(tab)` qui retourne un booléen indiquant si au moins un élément du tableau de nombres `tab` est strictement positif.

1.4.1 Corrigé de l'exercice 5

[32]:
```
def produit(tab):
    """
    Paramètres :
        tab est un tableau de type list contenant des  nombres de type int ou␣
 ↪float
    Précondition :
        tab non vide
    Valeur renvoyée :
        un tableau de type list contenant des  nombres de type int ou float
    Postcondition :
        le tableau renvoyé est constitué du produit
        de tous les termes de tab
    """
    assert len(tab) > 0, "le paramètre  doit être non vide"
    p = 1
    for e in tab:
        p = p * e
    return p

#Tests unitaires
from random import randint
from functools import reduce
for size in range(1, 5):
    for run in range(10):
        tab = [randint(-100, 100) for _ in range(size)]
        assert(produit(tab)) == reduce(lambda a, b: a * b,tab), "Echec sur {}".
 ↪format(tab)
print("Tests unitaires réussis")
```

```
Tests unitaires réussis
```

[36]:
```
def all_positive(tab):
    """
    Paramètres :
        tab est un tableau de type list contenant des  nombres de type int ou␣
 ↪float
    Précondition :
        tab non vide
    Valeur renvoyée :
        un booléen
    Postcondition :
        la valeur renvoyée est
            True si tous les éléments de tab sont positifs
            False sinon
    """
```

```
    assert len(tab) > 0, "le paramètre  doit être non vide"
    for e in tab:
        if e < 0:
            return False
    return True

#Tests unitaires
from random import randint
from functools import reduce
for size in range(1, 5):
    for run in range(10):
        tab = [randint(-100, 100) for _ in range(size)]
        assert(all_positive(tab)) == all([e >= 0 for e in tab]), "Echec sur {}".
 ↪format(tab)
print("Tests unitaires réussis")
```

```
Tests unitaires réussis
```

[37]:
```
def any_positive(tab):
    """
    Paramètres :
        tab est un tableau de type list contenant des  nombres de type int ou␣
 ↪float
    Précondition :
        tab non vide
    Valeur renvoyée :
        un booléen
    Postcondition :
        la valeur renvoyée est
            True si au moins un élément de tab est positif
            False sinon
    """
    assert len(tab) > 0, "le paramètre  doit être non vide"
    for e in tab:
        if e >= 0:
            return True
    return False

#Tests unitaires
from random import randint
from functools import reduce
for size in range(1, 5):
    for run in range(10):
        tab = [randint(-100, 100) for _ in range(size)]
        assert(any_positive(tab)) == any([e >= 0 for e in tab]), "Echec sur {}".
 ↪format(tab)
print("Tests unitaires réussis")
```

```
Tests unitaires réussis
```

1.5 Exercice 6

1. Écrire une fonction `tableau_aleatoire(n, a, b)` qui renvoie un tableau de `n` entiers tirés aléatoirement entre les entiers `a` et `b` inclus. On utilisera la fonction `randint` du module `random`.

```
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

2. Écrire une fonction `histo_echantillon(nbexp)` qui renvoie un tableau de taille 6 comptant le nombre d'occurences de chaque face numérotée de 1 à 6 sur un échantillon de `nbexp` lancers d'un dé cubique équilibré.

3. On donne ci-dessous une fonction qui prend en paramètre un tableau et renvoie un élément extrait au hasard du tableau. Elle permet par exemple de simuler un tirage sans remise d'une boule dans une urne.

```
from random import randint

def tirage_sans_remise(urne):
    """
    Paramètres :
        urne un tableau homogène type list
    Précondition :
        urne non vide
    Valeur renvoyée :
        un élément du même type que ceux dans urne
    Postcondition :
        l'élement renvoyée a été extrait aléatoirement de urne
        urne a été modifiée
    """
    return urne.pop(randint(0, .......))
```

- Compléter les pointillés pour respecter la spécification de la fonction.
- Écrire une fonction `echantillon_tirage_sans_remise` respectant la spécification ci-dessous. Les annotations de type sont des métadonnées optionnelles décrivant les types des paramètres et des valeurs renvoyées.

```
~~~python
def echantillon_tirage_sans_remise(urne : list, nbtirage : int) -> list:
    """
    Paramètres :
        urne un tableau homogène type list
        nbtirage un entier de type int
    Préconditions :
        nbtirage >= 0
        urne non vide
```

```
    Valeur renvoyée :
        un tableau d'entiers de type list
    Postcondition :
        le tableau renvoyé est un échantillon extrait
        aléatoirement de l'urne sans remise
    """
~~~
```

1.5.1 Corrigé de l'exercice 6

[50]:
```python
from random import randint

def tableau_aleatoire(n, a, b):
    """
    Paramètres :
       n, a et b trois entiers de type int
    Préconditions :
       n >= 0
       a <= b
    Valeur renvoyée :
       un tableau d'entiers de type list
    Postcondition :
        le tableau renvoyé contient  de n entiers tirés aléatoirement entre les␣
 ↪entiers a et b inclus
    """
    assert a <= b and n >= 0
    return [randint(a, b) for _ in range(n)]

def histo_echantillon(nbexp):
    """
    Paramètres :
       nbexp un entier de type int
    Préconditions :
       nbexp >= 0
    Valeur renvoyée :
       un tableau d'entiers de taille 6 de type list
    Postcondition :
        le tableau renvoyé contient le nombre d'occurences de chaque face de 1␣
 ↪à 6
        sur un échantillon de nbexp lancers de dé à 6 faces
    """
    assert nbexp >= 0, "la taille de l'échantillon doit être >= 0"
    histo = [0] * 6
    echantillon = tableau_aleatoire_aleatoire(nbexp, 1, 6)
    for e in echantillon:
        histo[e - 1] = histo[e - 1] + 1
    return histo
```

```python
def tirage_sans_remise(urne):
    """
    Paramètres :
        urne un tableau homogène type list
    Précondition :
        urne non vide
    Valeur renvoyée :
        un élément du même type que ceux dans urne
    Postcondition :
        l'élement renvoyée a été extrait aléatoirement de urne
        urne a été modifiée
    """
    return urne.pop(randint(0, len(urne) - 1))

def echantillon_tirage_sans_remise(urne : list, nbtirage : int) -> list:
    """
    Paramètres :
       urne un tableau homogène type list
       nbtirage un entier de type int
    Préconditions :
       nbtirage >= 0
       urne non vide
    Valeur renvoyée :
       un tableau d'entiers de type list
    Postcondition :
       le tableau renvoyé est un échantillon extrait
       aléatoirement de l'urne sans remise
    """
    assert len(urne) > 0 and nbtirage >= 0
    #tirage = []
    #for _ in range(nbtirage):
    #     tirage.append(tirage_sans_remise(urne))
    #return tirage
    return [tirage_sans_remise(urne) for _ in range(nbtirage)]
```

[55]:
```python
echantillon_tirage_sans_remise(list(range(7)), 4)
```

[55]:
```
[6, 1, 2, 3]
```

1.6 Exercice 7

- La fonction `ord` prend en paramètre un caractère de type `string` et renvoie son point de code dans le jeu de caractères Unicode.
- La fonction `chr` prend en paramètre un point de code Unicode et renvoie le caractère correspondant.

```
>>> ord('a')
```

```
97
>>> chr(97)
'a'
```

1. Construire un tableau `alphabet` qui contient toutes les lettres minuscules de l'alphabet romain.
2. Construire un tableau `consonne` qui contient toutes les consonnes dans `alphabet`.
3. Écrire une fonction `occurences(chaine)` qui prend en paramètre une chaîne de caractère et renvoie un tableau de taille 26 avec le nombre d'occurences dans `chaine` des 26 lettres de l'alphabet romain.

1.6.1 Corrigé de l'exercice 7

[80]:
```
alphabet = [chr(k) for k in range(ord('a'), ord('z') + 1)]
consonne = [c for c in alphabet if c not in 'aoeuiy']

def occurences(chaine):
    """
    Paramètres :
       chaine de type str
    Préconditions :
       chaine non vide ? pas forcément
    Valeur renvoyée :
       un tableau de 26 entiers de type list
    Postcondition :
       le tableau renvoyé contient le nombre d'occurences
       dans chaine des 26 lettres de l'alphabet romain.
    """
    histo = [0] * 26
    for c in chaine:
        index = ord(c)- ord('a')
        histo[index] = histo[index] + 1
    return histo

# Tests unitaires
assert occurences('search') == [1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,␣
↪0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
assert occurences('occurences') == [0, 0, 3, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1,␣
↪1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0]
assert occurences('') == [0] * 26
print('Tests unitaires réussis')
```

```
Tests unitaires réussis
```

1.7 Exercice 8

Exercice du manuel de NSI de Thibault Balabonski chez Ellipses

En mathématiques, la suite de Fibonacci est une séquence d'entiers définie ainsi : on part des

entiers 0 et 1 puis on construit à chaque fois l'entier suivant comme la somme des deux précédents :

$$0, 1, 1, 2, 3, 5....$$

Compléter la fonction `fibonacci(n)` ci-dessous pour qu'elle renvoie un tableau contenant les `n` premiers termes de la suite de Fibonacci avec `n` entier supposé supérieur ou égal à 2. Les deux assertions proposées doivent être vérifiées.

```
def fibonacci(n):
    """
    Paramètres :
       n un entier de type int
    Préconditions :
       n >= 0
    Valeur renvoyée :
       un tableau d'entiers de type list
    Postcondition :
       le tableau renvoyé contient tous les termes de la suite de Fibonacci
       d'indices compris entre 0 et n
    """

# Tests unitaires
f6 = fibonacci(6)
assert f6 == [0,1, 1, 2,3,5]
f30 = fibonacci(30)
assert f30[29] == 514229
print("Tests unitaires réussis !")
```

1.7.1 Corrigé de l'exercice 8

[62]:
```
def fibonacci(n):
    """
    Paramètres :
       n un entier de type int
    Préconditions :
       n >= 0
    Valeur renvoyée :
       un tableau d'entiers de type list
    Postcondition :
       le tableau contient tous les termes de la suite de Fibonacci
       d'indices compris entre 0 et n
    """
    assert n>= 0
    t = [0, 1]
    for k in range(n - 2):
        t.append(t[len(t)-2] + t[len(t)-1])
    return t
```

```python
# Test unitaires
f6 = fibonacci(6)
assert f6 == [0,1, 1, 2,3,5]
f30 = fibonacci(30)
assert f30[29] == 514229
print("Tests unitaires réussis !")
```

```
Tests unitaires réussis !
```

1.8 Exercice 9

- **Énoncé :**

Comme le disait le grand-père de Joseph Marchand, « Le plus beau voyage est celui qu'on n'a pas encore fait ». New York était dans la tête de Joseph depuis plusieurs années maintenant, et il a décidé aujourd'hui d'acheter son billet d'avion. Quelques secondes avant de cliquer sur le bouton « Acheter ! », son amie Haruhi lui envoie une liste de prix qu'elle a trouvés sur Internet. Joseph est curieux de voir si ces derniers sont moins chers que le billet qu'il s'apprêtait à acheter.

- **Entrée :**

Sur la première ligne le prix initial du billet de Joseph. Sur la deuxième ligne un entier N , correspondant au nombre de billets envoyés par Haruhi. La ligne suivante contient les N prix trouvés par Haruhi.

- **Sortie :**

Si Haruhi a trouvé au moins 3 prix strictement moins chers que celui de Joseph, affichez « ARNAQUE ! » pour l'avertir. Sinon « Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos ! ».

- Exemple 1 :
  - Entrée :

    ```
    570
    4
    495 1200 540 450
    ```

  - Sortie :

    ```
    ARNAQUE !
    ```

- Exemple 2:
  - Entrée :

    ```
    820
    5
    580 2000 970 1050 820
    ```

  - Sortie :

```
Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos !
```

1. Récupérer les fichiers textes `arnaque-exemple1.txt` et `arnaque-exemple2.txt` avec les entrées des deux exemples.
2. En s'inspirant de la méthode de lecture de fichier texte présentée précédemment, compléter la fonction ci-dessous pour qu'elle résolve le problème. Les deux assertions doivent être vérifiées.
3. Se créer un compte sur le site Prologin et soumettre sa solution au juge en ligne sur https://prologin.org/train/2019/qualification/arnaque_aerienne.

   Avant de soumettre, remplacer dans la fonction l'instruction `f = open(inputfile)` par `f = inputfile` et le code du programme principal par :

```
import sys
arnaque(sys.stdin)
```

`sys.stdin` est l'entrée standard de l'interpréteur Python.

4. Sur la même plateforme, résoudre cet autre problème du même type : https://prologin.org/train/2018/qualification/faites_place

```
def arnaque(inputfile):
    """Résolution du problème
    https://prologin.org/train/2019/qualification/arnaque_aerienne"""
    f = open(inputfile)  #ouverture du  fichier d'entrée
    # TO DO à compléter
    f.close()
    c = 0
    # TO DO à compléter
    if c >= 3:
        print("ARNAQUE !")
        return 1
    else:
        print("Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos !")
        return 0

# Tests unitaires
assert arnaque('arnaque-exemple1.txt') == 1
assert arnaque('arnaque-exemple2.txt') == 0
print("Tests unitaires réussis !")
```

1.8.1 Corrigé de l'exercice 9

[61]:
```
def arnaque(inputfile):
    """Résolution du problème
    https://prologin.org/train/2019/qualification/arnaque_aerienne"""
    f = open(inputfile)  #ouverture du  fichier d'entrée
    joseph = int(f.readline())
    N = int(f.readline())
    listeprix = [int(champ) for champ in f.readline().split()]
    f.close()
    c = 0
```

```
    for prix in listeprix:
        if prix < joseph:
            c = c + 1
    if c >= 3:
        print("ARNAQUE !")
        return 1
    else:
        print("Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos !")
        return 0

# Tests unitaires
assert arnaque('arnaque-exemple1.txt') == 1
assert arnaque('arnaque-exemple2.txt') == 0
print("Tests unitaires réussis !")
```

```
ARNAQUE !
Ok bon voyage, bisous, n'oublie pas de m'envoyer des photos !
Tests unitaires réussis !
```

[ ]:
```
import sys

def parseInt():
    """Lit une ligne de l'entrée standard, la découpe selon les espaces
    et la convertit en tableau d'entiers
    """
    return [int(c) for c in sys.stdin.readline().rstrip().split()]

longplage = parseInt()[0]
nbautres = parseInt()[0]
posautres = parseInt()
maxinfluence = posautres[0] - 1
for k in range(len(posautres) - 1):
    maxinfluence = max(maxinfluence, (posautres[k+1] - posautres[k])//2)
maxinfluence = max(maxinfluence, longplage - posautres[-1] - 1)
sys.stdout.write(str(maxinfluence))
```

1.9 Exercice 10

Écrire une fonction `maximum_intervalle(t, n)` qui prend en paramètres un tableau d'entiers `t` et un entier `n` supposé inférieur ou égal à la longueur de `t` et qui retourne la somme maximale sur tous les sous-tableaux de longueur `n` inclus dans `t` .

```
>>> maximum_intervalle([10,2,8,4,7,5], 3)
16
```

1.9.1 Corrigé de l'exercice 10

[70]:
```python
def maximum_intervalle(t, n):
    """
    Paramètres :
       t un tableau de type list contenant des entiers de  type  int
       n un entier de type int
    Préconditions :
       len(t) >= 0
       0 <= n <= len(t)
    Valeur renvoyée :
       un entier de type int
    Postcondition :
       l'entier renvoyé est la somme maximale sur tous les sous-tableaux de
       de longueur n
    """
    assert len(t) >= 0 and 0 <= n <= len(t)
    #initialisation de la somme
    s = 0
    for i in range(n):
        s = s + t[i]
    smax = s
    for j in range(n, len(t)):
        s = s - t[j - n] + t[j]
        if s > smax:
            smax = s
    return smax


# Tests unitaires
assert maximum_intervalle([4,1,10,2,8,5], 3) == 20
assert maximum_intervalle([10,2,8,4,1,5], 3) == 20
assert maximum_intervalle([4,1,5, 10,2,8], 3) == 20
print("Tests unitaires réussis")
```

```
Tests unitaires réussis
```