Exercices sur les tableaux (correction)

ACTIVITÉ 1

Exercice 1

Créer une fonction `somme_pda(age_min, age_max))` qui renvoie le nombre total de personne en France ayant entre `age_min` ans (inclu)) et `age_max` ans (exclu)).

Pour cela, on s'appuiera sur les données 2021 de l'INSEE dont les valeurs sont données ci-dessous. Le tableau `pda` contient le nombre de personnes. Chaque indice du tableau correspond à un âge. Donc la case `pda[10]` contient le nombre de personnes ayant 10 ans. Le tableau `pda` contient 100 valeurs.

```
pda = [691754, 714585, 723589, 741105, 761638, 782397, 806296,
       857412, 846079, 853732, 844444, 859144, 843717, 839171,
       825068, 771160, 758647, 732962, 739197, 732521, 715066,
       796258, 802788, 815670, 818399, 832977, 832461, 825974,
       881060, 833394, 813784, 816353, 795349, 820771, 861730,
       899550, 886276, 875462, 871272, 891426, 892392, 899362,
       848042, 841009, 820136, 812492, 804014, 793523, 786152,
       773191, 754577, 747317, 725950, 679274, 507127, 490957,
       383970, 392579, 373969, 355308, 341153, 314805, 301041,
       198098, 158712, 134827, 108837, 88250, 69403, 52436, 388

def somme_pda(age_min, age_max)):
    ...
```

Tests et exemples d'usage :

```
>>> print(somme_pda(0, 1))
691754
>>> print(somme_pda(0, 100))
67387330
>>> print(somme_pda(50, 67))
14519530
```

[2]:
```
pda = [691754, 714585, 723589, 741105, 761638, 782397, 806296, 813727, 830361, 836626,
       857412, 846079, 853732, 844444, 859144, 843717, 839171, 830450, 823756, 824021,
       825068, 771160, 758647, 732962, 739197, 732521, 715066, 719576, 756910, 774944,
       796258, 802788, 815670, 818399, 832977, 832461, 825974, 817778, 865383, 871959,
       881060, 833394, 813784, 816353, 795349, 820771, 861730, 906113, 924323, 919896,
       899550, 886276, 875462, 871272, 891426, 892392, 899362, 886362, 856619, 855441,
       848042, 841009, 820136, 812492, 804014, 793523, 786152, 768512, 775095, 751752,
       773191, 754577, 747317, 725950, 679274, 507127, 490957, 471618, 430510, 376688,
       383970, 392579, 373969, 355308, 341153, 314805, 301041, 269011, 253855, 222346,
       198098, 158712, 134827, 108837, 88250, 69403, 52436, 38806, 27854, 27497]
```

[4]:
```
def somme_pda(age_min, age_max)):
    n = 0
    for age in range(age_min, age_max)):
        n += pda[age]

    return n

print(somme_pda(0, 1))))
print(somme_pda(0, 100))))
print(somme_pda(50, 67))))
```

691754
67387330
14519530

ACTIVITÉ 2

Exercice 2

Créer une fonction somme_pda(age_min, age_max)) plus robuste qui : - qui renvoie le nombre total de personne en France ayant entre age_min ans (inclu)) et age_max ans (exclu)) ; - qui **n'échoue pas** si l'utilisateur spécifie un age_max arbitrairement grand.

Pour cela, on s'appuiera sur les données 2021 de l'INSEE dont les valeurs sont données ci-dessous. Le tableau `pda` contient le nombre de personnes. Chaque indice du tableau correspond à un âge. Donc la case `pda[10]` contient le nombre de personnes ayant 10 ans. Le tableau `pda` contient 100 valeurs.

```
pda = [691754, 714585, 723589, 741105, 761638, 782397, 806296,
       857412, 846079, 853732, 844444, 859144, 843717, 839171,
       825068, 771160, 758647, 732962, 739197, 732521, 715066,
       796258, 802788, 815670, 818399, 832977, 832461, 825974,
       881060, 833394, 813784, 816353, 795349, 820771, 861730,
       899550, 886276, 875462, 871272, 891426, 892392, 899362,
       848042, 841009, 820136, 812492, 804014, 793523, 786152,
       773191, 754577, 747317, 725950, 679274, 507127, 490957,
       383970, 392579, 373969, 355308, 341153, 314805, 301041,
       198098, 158712, 134827, 108837, 88250, 69403, 52436, 388

def somme_pda(age_min, age_max)):
    ...
```

Tests et exemples d'usage :

```
>>> print(somme_pda(0, 1))
691754
>>> print(somme_pda(1, 0))
0
>>> print(somme_pda(0, 100))
67387330
>>> print(somme_pda(0, 101))
67387330
>>> print(somme_pda(0, 150))
```

```
67387330
>>> print(somme_pda(50, 67))
14519530
>>> print(somme_pda(50, 100))
26884855
>>> print(somme_pda(50, 242))
26884855
```

[6]:
```
pda = [691754, 714585, 723589, 741105, 761638, 782397, 806296, 813727, 830361, 836626,
       857412, 846079, 853732, 844444, 859144, 843717, 839171, 830450, 823756, 824021,
       825068, 771160, 758647, 732962, 739197, 732521, 715066, 719576, 756910, 774944,
       796258, 802788, 815670, 818399, 832977, 832461, 825974, 817778, 865383, 871959,
       881060, 833394, 813784, 816353, 795349, 820771, 861730, 906113, 924323, 919896,
       899550, 886276, 875462, 871272, 891426, 892392, 899362, 886362, 856619, 855441,
       848042, 841009, 820136, 812492, 804014, 793523, 786152, 768512, 775095, 751752,
       773191, 754577, 747317, 725950, 679274, 507127, 490957, 471618, 430510, 376688,
       383970, 392579, 373969, 355308, 341153, 314805, 301041, 269011, 253855, 222346,
       198098, 158712, 134827, 108837, 88250, 69403, 52436, 38806, 27854, 27497]
```

[8]:
```
def somme_pda(age_min, age_max):
    n = 0
    if age_max > len(pda):
        age_max = len(pda)

    for age in range(age_min, age_max):
        n += pda[age]

    return n
```

[11]:
```
print(somme_pda(0, 1))
print(somme_pda(0, 100))
print(somme_pda(0, 101))
print(somme_pda(0, 150))
print(somme_pda(50, 67))
print(somme_pda(50, 100))
print(somme_pda(50, 242))
```

```
691754
67387330
67387330
67387330
14519530
26884855
26884855
```

ACTIVITÉ 3

Exercice 3

Écrire une fonction `occurences(val, tab)` qui renvoie le nombre d'occurrences de la valeur `val` dans le tableau `tab`.

Tests et exemples d'usage :

```
>>> occurences(0, [0, 1, 1, 3, 1])
1
>>> occurences(1, [0, 1, 1, 3, 1])
3
>>> occurences("a", ["abc", "a", "b", "c"])
1
```

[12]:
```
def occurences(val, tab):
    n = 0
    for i in range(0, len(tab)):
        if tab[i] == val:
            n = n + 1

    return n
```

[15]:
```
print(occurences(0, [0, 1, 1, 3, 1]))
print(occurences(1, [0, 1, 1, 3, 1]))
print(occurences("a", ["abc", "a", "b", "c"]))
```

```
1
3
1
```

ACTIVITÉ 4

Exercice 4

Écrire un programme qui construit un tableau de 100 entiers tirés au hasard entre 1 et 1000, puis qui l'affiche.

Attention : Pour les besoins du juge en ligne (= tests automatiques), il faut commencer votre programme par :

```
from random import *
seed(42)
...
```

Tests et exemples d'usage :

```
[655, 115,... , 167, 380]
```

[16]:
```
from random import *
seed(42)

tab = [0] * 100
for i in range(100):
    tab[i] = randint(1, 1000)
print(tab)
```

```
[655, 115, 26, 760, 282, 251, 229, 143, 755, 105, 693, 759,␣
 ↪914, 559, 90, 605,
433, 33, 31, 96, 224, 239, 518, 617, 28, 575, 204, 734, 666,␣
 ↪719, 559, 430, 226,
460, 604, 285, 829, 891, 7, 778, 826, 164, 715, 433, 349,␣
 ↪285, 160, 221, 981,
782, 345, 105, 95, 390, 100, 368, 868, 353, 619, 271, 827,␣
 ↪45, 748, 471, 550,
128, 997, 945, 388, 81, 566, 301, 850, 644, 634, 907, 883,␣
 ↪371, 592, 197, 722,
72, 47, 678, 234, 792, 297, 82, 876, 239, 888, 104, 390, 285,␣
 ↪465, 651, 855,
374, 167, 380]
```

ACTIVITÉ 5

Exercice 5

Écrire une fonction maxi_tab(t) qui prend en paramètre un tableau t et renvoie la valeur maximale de ce tableau.

Tests et exemples d'usage :

```
>>> maxi_tab([1, 2, 3])
3
>>> maxi_tab([10, 11, 3, 4])
11
```

[18]:
```
def maxi_tab(t):
    maxi = t[0]
    for i in range(1, len(t)):
        if t[i] > maxi:
            maxi = t[i]
    return maxi

print(maxi_tab([1, 2, 3]))
print(maxi_tab([10, 11, 3, 4]))
```

```
3
11
```

ACTIVITÉ 6

Exercice 6.a

Créer un programme qui tire au hasard mille entiers entre 1 et 10 et affiche ensuite le nombre de fois que chaque nombre a été tiré. Afficher le résultats sous la forme `"le nombre ... a été tiré ... fois"`.

Remarque : pour des raisons de juge en ligne, commencer votre code par :

```
from random import *
seed(42)
...
```

Tests et exemples d'usage :

```
le nombre 1 a été tiré 91 fois
le nombre 2 a été tiré 108 fois
...
...
le nombre 10 a été tiré 95 fois
```

[20]:
```
from random import *
seed(42)

effectifs = [0] * 10

for i in range(1000):
    n = randint(1, 10)
    effectifs[n - 1] += 1

for i in range(len(effectifs)):
        print("le nombre", i+1, "a été tiré", effectifs[i], "fois")
```

```
le nombre 1 a été tiré 91 fois
le nombre 2 a été tiré 108 fois
le nombre 3 a été tiré 85 fois
le nombre 4 a été tiré 116 fois
le nombre 5 a été tiré 113 fois
le nombre 6 a été tiré 82 fois
le nombre 7 a été tiré 102 fois
le nombre 8 a été tiré 105 fois
le nombre 9 a été tiré 103 fois
le nombre 10 a été tiré 95 fois
```

ACTIVITÉ 7

Exercice 6.b

Vous allez modifier légèrement le code de l'exercice précédent afin de pouvoir le tester plus efficacement avec le juge en ligne.

Pour cela :

1. demander à l'utilisateur de saisir un nombre entier que vous affecterez à la variable `gen`.
2. Copier/coller ensuite votre code de l'exercice précédent.
3. Remplacer l'instruction `seed(42)` par `seed(gen)`.

Normalement, votre programme devrait commencer de la façon suivante :

```
gen = int(input("saisir une valeur pour le générateur aléatoir
from random import *
seed(gen)
...
```

Ainsi un utilisateur qui saisi le nombre `42` devrait voir s'afficher exactement le même résultat que l'exercice précédent.

Le juge en ligne peut désormais tester de multiples façon votre programme :)

Tests et exemples d'usage :

```
>>> saisir une valeur pour le générateur aléatoire : 42
le nombre 1 a été tiré 91 fois
le nombre 2 a été tiré 108 fois
...
```

```
le nombre 9 a été tiré 103 fois
le nombre 10 a été tiré 95 fois
```

[23]:
```python
gen = int(input("saisir une valeur pour le générateur aléatoire : "))
from random import *
seed(gen)

effectifs = [0] * 10

for i in range(1000):
    n = randint(1, 10)
    effectifs[n - 1] += 1

for i in range(len(effectifs)):
    print("le nombre", i+1, "a été tiré", effectifs[i], "fois")
```

```
le nombre 1 a été tiré 91 fois
le nombre 2 a été tiré 108 fois
le nombre 3 a été tiré 85 fois
le nombre 4 a été tiré 116 fois
le nombre 5 a été tiré 113 fois
le nombre 6 a été tiré 82 fois
le nombre 7 a été tiré 102 fois
le nombre 8 a été tiré 105 fois
le nombre 9 a été tiré 103 fois
le nombre 10 a été tiré 95 fois
```

ACTIVITÉ 8

Exercice 7

En mathématiques, la très célèbre suite de **Fibonacci** est une séquence infinie d'entiers définie de la façon suivante : on part des deux entiers 0 et 1 puis on construit à chaque fois l'entier suivant comme la somme des deux entiers précédents.

$$0, 1, 1, 2, 3, 5, \ldots$$

Écrire un programme qui construit puis affiche un tableau contenant les 30 premiers termes de la suite.

Tests et exemples d'usage : “‘python [0, 1, 1, 2, 3, 5,. . . , 514229]

[24]:
```
tab = [0] * 30

tab[0] = 0
tab[1] = 1

for i in range(2, 30):
    tab[i] = tab[i-2] + tab[i-1]

print(tab)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610,␣
↪987, 1597, 2584,
4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393,␣
↪196418, 317811, 514229]
```

ACTIVITÉ 9

Exercice 8

Écrire une fonction `copie(tab)` qui prend en paramètre un tableau `tab` et renvoie une copie de ce tableau.

Tests et exemple d'usage :

```
>>> u = []              # u est un tableau vide
>>> t = [1, 2, 3]
>>> u = copie(t)
>>> print(u)            # vérifier que u contient les mêmes valeur
[1, 2, 3]
>>> t[2] = 7            # vérifier que lorsqu'on modifie t
```

```
>>> print(t)          #   alors u n'est PAS modifié par effets d
[1, 2, 7]
>>> print(u)
[1, 2, 3]

return tab_2
```

[30]:
```
def copie(tab):
    n = len(tab)
    tab_2 = [0] * n

    for i in range(n):
        tab_2[i] = tab[i]

    return tab_2
```

[31]:
```
u = []          # u est un tableau vide
t = [1, 2, 3]
u = copie(t)
print(u)        # vérifier que u contient les mêmes valeurs que t
```

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

[32]:
```
t[2] = 7        # vérifier que lorsqu'on modifie t
print(t)
```

```
[1, 2, 7]
```

[33]:
```
print(u)
```

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

ACTIVITÉ 10

Exercice 9

Écrire une fonction `ajout(val, tab)` qui renvoie un nouveau tableau contenant d'abord tous les éléments de `tab` puis la valeur `val`.

Tests et exemples :

```
>>> print( ajout(42, []) )
[42]

>>> print( ajout(10, [5, 4, 3, 2, 1]) )
[5, 4, 3, 2, 1, 10]

>>> u = []
>>> t = [1, 2, 3]
>>> u = ajout(4, t)
>>> print(u)          # u est une copie de t avec 4 ajouté à la
[1, 2, 3, 4]
>>> print(t)          # t inchangé
[1, 2, 3]
```

[34]:
```
def ajout(val, tab):
    n = len(tab)
    tab_2 = [0] * (n + 1)
    for i in range(n):
        tab_2[i] = tab[i]
    tab_2[n] = val
    return tab_2
```

[35]:
```
u = []
t = [1, 2, 3]
u = ajout(4, t)
print(u)        # u est une copie de t avec 4 ajouté à la fin
print(t)        # t inchangé
print( ajout(10, [5, 4, 3, 2, 1]) )
```

```
[1, 2, 3, 4]
[1, 2, 3]
[5, 4, 3, 2, 1, 10]
```