Corrigé de la feuille d'exercices 3
Épreuves types - banque PT

Exercice 1

1. Fonction `comptage`.

```
def comptage(L,N):
    P = [0] * N
    for n in L:
        P[n] += 1
    return P
```

2. Fonction `tri`.

```
def tri(L,N):
    P = comptage(L,N)
    i = 0
    for j in range(N):
        x = P[j]
        for k in range(x):
            L[i] = j
            i += 1
    return L
```

3. La fonction `randint(0,5)` du module `random` retourne un nombre entier pseudo aléatoire entre 0 et 5 inclus.
On pourrait aussi utiliser la fonction `random()` qui retourne un float dans $[0, 1[$ (loi pseudo-uniforme), et `int(6*random())`.

```
In [1]: from random import randint
In [2]: L = [ randint(0,5) for i in range(20) ]
In [3]: tri(L, 6)
Out[3]:
[0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 3, 3, 5, 5, 5, 5, 5, 5, 5]
```

4. Complexité de `comptage` :

N opérations pour créer le tableau P ; parcours de `L` par la boucle `for` : `len(L)` opérations : incrémentations d'un élément de P (l'écriture d'un élément dans une liste est en $O(1)$).

Pour un tableau de n éléments , constitués de valeurs dans $[\![0, N]\!]$, la complexité temporelle de `comptage()` est en $\Theta(\max(n, N))$ (et spatiale en $O(N)$).

Complexité de `tri` :

C'est la complexité de comptage ($\max(n, N)$) plus la complexité de la reconstitution de la liste `L` triée :

Reconstitution de `L` triée : de l'ordre de :

$$P[0] + P[1] + \cdots + P[N-1] = n \quad \text{opérations}$$

(puisque la somme du nombre d'occurrences des éléments apparaissant dans `L` égale son nombre d'éléments au total).

Reconstitution de `L` triée de complexité : $\Theta(n)$.

Ainsi, l'algorithme est de complexité : $\Theta(\max(N, n))$ (où n est la longueur de la liste L à trier).

L'algorithme est intéressant seulement lorsque les valeurs sont des entiers positifs peu étalés (N peu grand devant n). Pour des entiers uniformément bornés, il est intéressant car de complexité linéaire.
Autrement, lorsque $N >> n$, il faut l'éviter et privilégier d'autres algorithmes de tri.

Exercice 2

1. En sommant tous les éléments d'une matrice magique de deux façons différentes, on obtient :

$$n \times s = \sum_{k=1}^{n^2} k = \frac{n^2(n^2+1)}{2} \implies s = \frac{n(n^2+1)}{2} .$$

2. Fonction `EstMagique`.

```
def EstMagique(T):
    n = len(T)
    # 1 : tester que tous les entiers de 1 à n^2 apparaissent
    L = [False] * n**2
    for i in range(n):
        for j in range(n):
            x = T[i][j]
            if not(x == int(x) and 0 <x<= n**2) or L[x-1]:
                return False
            L[x-1] = True
    # 2 : tester que la somme de chaque ligne/colonne est s
    s = n*(n**2+1)/2
    for i in range(n):
        S1 = S2 = 0
        for j in range(n):
            S1 += T[i][j]
            S2 += T[j][i]
        if S1 != s or S2 != s:
            return False
    return True
```

Test : on utilise la fonction `array` de `numpy`.

```
In [2]: from numpy import array

In [2]: A = array([[4,9,2],[3,5,7],[8,1,6]])

In [3]: B = array([[1,8,2],[4,5,7],[6,9,3]])

In [4]: EstMagique(A)
Out[4]: True

In [5]: EstMagique(B)
Out[5]: False
```

3. La matrice d'ordre 3 obtenue par cette méthode est :

$$\begin{pmatrix} 3 & 5 & 7 \\ 8 & 1 & 6 \\ 4 & 9 & 2 \end{pmatrix}$$

4. Fonction `Magique`.

```
from numpy import zeros

def Magique(n):
    if n%2==0:
        return
    M = zeros((n,n),dtype=int)
    # Ou bien : M= [[0]*n for k in range(n)]
    i = j = n//2
    M[i][j] = 1
    for k in range(2,n**2+1):
        i = (i+1)%n
        j = (j+1)%n
        if M[i][j] == 0:
            M[i][j] = k
        else:
            i = (i+1)%n
            j = (j-1)%n
            M[i][j] = k
    return M
```

Exercice 3

1. Fonction `suivant` :

```
def suivant(E,p):
    n = len(E)
    personne = True
    while personne:
        p = (p+1) % n
        personne = E[p]
    return p
```

2. Simulation du cas $n = 5$.

```
E = [False] * 5
p = 4
for boucle in range(4):
    p = suivant(E,p)
    p = suivant(E,p)
    E[p] = True
    print(E)
```

L'exécution affiche, comme attendu :

```
[False, True, False, False, False]
[False, True, False, True, False]
[True, True, False, True, False]
[True, True, False, True, True]
```

3. Fonction principale, `reste`. Au sein d'une boucle, on appelle deux fois la fonction `suivant` avant de changer la valeur de l'élément trouvé dans la liste.

```
def reste(n):
    E = [False] * n
    p = n-1
    for boucle in range(n-1):
        p = suivant(E,p)
        p = suivant(E,p)
        E[p] = True
    return suivant(E,p)
```

4. Code pour l'affichage.

```
for n in range(2,141):
    print("n =",n,"Dernier restant :",reste(n))
```

On remarque que le reste vaut 0 pour toutes les puissances de 2 : 2, 4, 16, 32, 64, 128, puis s'incrémente de 2 à chaque nouvelle itération.

En notant N la plus grande puissance de 2 inférieure ou égale à n, c'est-à-dire : $N = 2^{\lfloor \log_2(n) \rfloor}$, on conjecture que le dernier restant parmi n personnes vaut $2 \times (n - N)$, soit :

$$2 \times \left(n - 2^{\lfloor \log_2(n) \rfloor}\right) .$$

5. En admettant la conjecture faite :

n	$N = 2^{\lfloor \log_2(n) \rfloor}$	$(n - N)$	Dernier restant
256	256	0	0
513	512	1	2
1023	512	511	1022
1041	1024	17	34

6. Fonction `reste2`.

```
from math import log2

def reste2(n):
    N = 2**int(log2(n))
    return (n-N)*2
```

Vérification :

```
In [2]: reste(5104), reste2(5104))
(2016, 2016)
```

La conjecture est vérifiée pour $n = 5104$.

Exercice 4

1. Déclaration des variables globales de type liste, `MC` et `ML`.

```
MC = ['avril','juin','septembre','novembre']
ML = ['janvier','mars','mai','juillet','aout','octobre',\
                                                'decembre']
```

2. Fonction `estBissextile`.

```
def estBissextile(an):
    if an%400 == 0 or (an%4 == 0 and an%100 != 0):
        return True
    else:
        return False
```

Fonction `longueurmois`.

```
def longueurmois(ms,an):
    if ms == 'fevrier':
        if estBissextile(an):
            return 29
        else:
            return 28
    else:
        if ms in MC:
            return 30
        elif ms in ML:
            return 31
```

3. Fonction `valide`.

```
def valide(jr,ms,an):
    if type(jr) != int or type(ms) != str or type(an) != int:
        return False
    if an <= 1582:
        return False
    if ms == 'fevrier' or ms in MC or ms in ML:
        if 0 < jr <= longueurmois(ms,an):
            return True
    return False
```

4. On commence par écrire une fonction qui prend en arguments deux dates, et les renvoie ordonnées par ordre chronologique.

```
def ordre(date1,date2):
    mois = ['janvier', 'fevrier', 'mars', 'avril', 'mai',\
     'juin', 'juillet', 'aout', 'septembre', 'octobre', \
     'novembre', 'decembre']
    mois1, mois2 = mois.index(date1[1]), mois.index(date2[1])
    if date2[2] < date1[2]:
        date1, date2 = date2, date1
    elif date2[2] == date1[2]:
        if mois2 < mois1:
            date1, date2 = date2, date1
        elif mois2 == mois1:
            if date2[0] < date1[0]:
                date1, date2 = date2, date1
    return date1, date2
```

Fonction `nab` qui renvoie le nombre "29 février" entre deux dates. Elle utilise la fonction `ordre` pour que `date1` soit antérieure à `date2`.

```
def nab(date1,date2):
    date1, date2 = ordre(date1,date2)
```

```
    an1, an2 = date1[2], date2[2]
    B = 0
    for an in range(an1+1,an2):
        if estBissextile(an):
            B += 1
    if estBissextile(an1):
        if date1[1] in ('janvier','fevrier'):
            if an2 > an1\
                 or (date2[1] not in ('janvier','fevrier')):
                B += 1
    if estBissextile(an2):
        if date2[1] not in ('janvier','fevrier'):
            if an1 < an2\
                 or (date1[1] in ('janvier','fevrier')):
                B += 1
    return B
```

5. Fonction `jours`.
Comme c'est la fonction principale (par exemple pour réaliser un ¡¡ calendrier perpétuel ¿¿) elle commence par vérifier que les deux dates sont valides. Elle utilise ensuite la fonction `ordre` pour que `date1` soit antérieure à `date2`.
On compte d'abord le nombre de jours des années pleines (du 1er janvier au 31 décembre), puis on lui ajoute le nombre de jours résiduels les première et dernière années.

```
def jours(date1,date2):
    # Verification de la validite des dates
    if not(valide(date1[0], date1[1], date1[2])\
         and valide(date2[0], date2[1], date2[2])):
        print('Dates invalides')
        return
    # Ordre chronologique
    date1, date2 = ordre(date1,date2)
    # Nombre de jours les annees pleines
    J = (date2[2]-date1[2]-1)*365 + nab(date1,date2)
    # Liste des mois
    mois = ['janvier', 'fevrier', 'mars', 'avril', 'mai',\
        'juin', 'juillet', 'aout', 'septembre', 'octobre',\
                'novembre', 'decembre']
    # Calcul des jours residuels
    mois1 = mois.index(date1[1])
    mois2 = mois.index(date2[1])
    for i in range(mois2):
        nomMois = mois[i]
        if nomMois == 'fevrier':
            J += 28
        elif nomMois in MC:
            J += 30
        else:
            J += 31
    J += date2[0]
    for i in range(mois1, 12):
        nomMois = mois[i]
        if nomMois == 'fevrier':
            J += 28
        elif nomMois in MC:
            J += 30
        else:
            J += 31
    J -= date1[0]
    return J
```