

**CCP 2017 TSI – corrigé**

1.

```
import os
import scipy.misc as scm
os.chdir("C:/CCP")
image_terrain = scm.imread("stade.bmp")
scm.imshow(image_terrain)
```

2.

```
dim_large, dim_long = image_terrain.shape
print("%d x %d"%(dim_long, dim_large))
```

3.

```
def coul(image):
    hauteur, largeur = image.shape
    x = largeur//2+2
    y = hauteur//2
    coul_ter = image[y,x]
```

4. La variable coul\_ter est un tableau. Un pixel contient trois valeurs codées sur 8 bits, il est donc codé sur 24 bits.

5.

```
def maillot():
    coul_ter = coul(image_terrain)
    coul_blanc = [255,255,255]
    return [coul_ter, coul_blanc]
```

6.

```
def filtrer1(filtreA, matB):
    nA = filtreA.shape[0]
    nb_ligneB = matB.shape[0]
    nb_colonneB = matB.shape[1]
    C = matB.copy()
    bordure = nA//2
    for i in range(bordure, nb_ligneB-bordure):
        for j in range(bordure, nb_colonneB-bordure):
            Bij = matB[i-bordure:i+bordure+1, j-bordure:j+bordure+1]
            C[i, j] = sum(dot(Bij, filtreA))
    return C
```

7.

```
def filtrer(filtreA, matB):
    C = matB.copy()
    for i in range(3):
        C[:, :, i] = filtrer1(filtreA, matB[:, :, i])
    return C
```

8.

```
def matriceFlouGaussien(taille, sigma):
    """
    taille : taille de la matrice (impair)
    sigma : écart-type (déviations standard)
    retourne un niveau gaussien
    """
    mat = zeros([taille, taille])
    taille = taille//2
    for x in range(-taille, taille+1):
        for y in range(-taille, taille+1):
```

```

    mat[y+taille, x+taille] = exp(-(x**2+y**2)/(2*sigma))
    return mat/sum(mat)

```

9.

```

def FloutageGaussien(tabPix, taille, sigma):
    return filtrer(matriceFlouGaussien(taille, sigma), tabPix)

```

10.

```

x = []
y_plaR = []
for i in range(len(resultats)):
    x.append(resultats[i,0])
    y_plaR.append(resultats[i,3])

```

11.

```

barre = []
abscisse = []
for i in range(len(x)):
    b = []
    a = []
    p = 0
    while p < y_plaR[i]:
        a.append(x[i])
        b.append(p)
        p+=0.1
    barre.append(b)
    abscisse.append(a)

```

12.

```

def minMaxMoy(valeurs):
    m = M = valeurs[0]
    S = 0
    for v in valeurs:
        S+=v
        if v>M:
            M=v
        if v<m:
            m=v
    return [m,M,S/len(valeurs)]

```

13.

```

| SELECT Nom FROM Joueurs WHERE Age>23 AND VMA>13;

```

14.

```

| SELECT Clubs.Nom FROM Clubs JOIN Joueurs ON Clubs.Id_Club = Joueurs.id_Club
| WHERE Joueurs.Salaire > 30000;

```

```

| SELECT COUNT(*) FROM Clubs JOIN Joueurs ON Clubs.Id_Club = Joueurs.id_Club
| WHERE Clubs.Nom = "Stade Toulousain" AND Joueurs.Poste="Talonneur";

```

15.

```

| SELECT 100*SUM(Joueurs.Salaire)/Clubs.BudgetTotal FROM Clubs JOIN Joueurs ON Clubs.Id_Club
| GROUP BY Clubs.Id_Club;

```

16. La variable monequipe est une liste de listes.

17.

```
def echange(l, i, j):
    """ Échange 2 valeurs d'une liste """
    l[i], l[j] = l[j], l[i]

def tri_1(liste, critere):
    """ Trie la liste en fonction du critère choisi """
    for i in range(len(liste)): # Il manque le len()
        mini = i
        for j in range(i+1, len(liste)):
            if liste[j][critere] < liste[mini][critere]: # mauvais sens de l'inégalité
                mini = j
        echange(liste, i, mini)
    return liste
tri_1(monequipe, 5)
print(monequipe)
```

18. Dans le pire des cas, la liste est triée à l'envers et la boucle intérieure est exécutée  $n - i$  fois. La complexité est donc :

$$C(n) = n - 1 + n - 2 + \dots + 1 = \frac{n(n-1)}{2} = O(n^2)$$

19.

```
def tri_2(L, val, i, j):
    """ Il s'agit d'une version du tri rapide
    La fonction segmente sert à partitionner
    la liste et renvoie la position du pivot
    i est l'indice de gauche de la liste à trier,
    j est l'indice de droite
    val est le critère de tri
    """
    # On ne trie que si la liste n'est pas vide : i < j
    if i < j:
        k = segmente(L, val, i, j) # Partitionne la liste entre i et j
        tri_2(L, val, i, k-1) # Tri rapide de la partie gauche
        tri_2(L, val, k+1, j) # Tri rapide de la partie droite
    return L
```

Cette fonction tri\_2 est récursive. Pour avoir le nombre d'appels récursifs, on peut par exemple faire :

```
def tri_2(L, val, i, j, n):
    """ Il s'agit d'une version du tri rapide
    La fonction segmente sert à partitionner
    la liste et renvoie la position du pivot
    i est l'indice de gauche de la liste à trier,
    j est l'indice de droite
    val est le critère de tri
    """
    n += 1
    # On ne trie que si la liste n'est pas vide : i < j
    if i < j:
        k = segmente(L, val, i, j) # Partitionne la liste entre i et j
        n = tri_2(L, val, i, k-1, n) # Tri rapide de la partie gauche
        n = tri_2(L, val, k+1, j, n) # Tri rapide de la partie droite
    return n
```

20. Pour trier la liste des joueurs en fonction du poids des joueurs, on utilisera l'instruction suivante :

```
tri_2(resultats, 3, 0, len(resultats)-1)
```

21.  $74,25 = 64 + 8 + 2 + \frac{1}{4} = 2^6 + 2^3 + 2^1 + 2^{-2} = 1\,001\,010,01_2$

22. L'énoncé est loin de donner toutes les informations nécessaires pour coder ce nombre au format IEEE754, on a :

— 0 pour le bit de signe (nombre positif);

- 101 001 000 000 000 000 000 pour la mantisse (le bit 1 le plus à gauche est implicite) ce qui correspond au nombre binaire 1,001 010 01 ;
- L'exposant est alors  $e = 6$  soit avec le biais on obtient  $e = 6 + 127 = 133 = 10000101_2$ .

On obtient alors le nombre 01000010100101001000000000000000.

23. 3000 joueurs ayant chacun 3 caractéristiques codées sur 32 bits (4 octets), cela donne  $3000 * 3 * 4/1000 = 36$  ko de données. Le format simple précision occupe moins d'espace mémoire, mais stocke les nombres avec une précision plus faible.
- 24.

```
def liste_temps(pas, tmax):
    temps = [0]
    while(temps[-1]<tmax-pas):
        temps.append(temps[-1]+pas)
```

25. On demande la solution analytique de l'équation différentielle :

$$\tau \frac{dv}{dt} + v(t) = K_c U_0$$

C'est une équation différentielle linéaire d'ordre 1 à coefficients constants. On trouve sans problème, en utilisant  $v(0) = 0$  :

$$v(t) = K_c U_0 (1 - \exp(-t/\tau))$$

On écrit alors la fonction suivante :

```
def vitesse(k, tau, u, temps):
    v = []
    for t in temps:
        v.append(k*u*(1-exp(-t/tau)))
    return v
```

- 26.

```
def ordre1_euler(k, tau, u, temps):
    v = [0]
    tp = temps[0] # Vitesse précédente
    vp = v[0]      # temps précédent
    for t in temps[1:]:
        v.append(vp + (t-tp)*(k*u-vp)/tau)
        vp = v
        tp = t
    return v
```

- 27.

```
for pas in [0.2, 0.4, 0.6]:
    temps = liste_temps(pas, tmax)
    vitesse = ordre1_euler(k, tau, u, temps)
```

- 28.

```
import matplotlib.pyplot as plt
for pas in [0.2, 0.4, 0.6]:
    temps = liste_temps(pas, tmax)
    vitesse = ordre1_euler(k, tau, u, temps)
    plt.plot(temps, vitesse)
plt.show()
```