1.
```
import os
os.chdir("C :/CCP/")
import scipy.misc as scm
image_terrain = scm.imread("stade.bmp")
imshow(image_terrain)
```

2.
```
dim_long = image_terrain.shape[1]
dim_larg = image_terrain.shape[0]

print(dim_long, "x", dim_large)
```

3.
```
def coul(image) :
    coul_ter=image[image.shape[0]//2][image.shape[1]//2+4]
    return coul_ter
```

Remarque : la mention dans l'énoncé de cette variable `coul_ter` est étrange, on peut très bien retourner directement la valeur sans la stocker dans une variable...

4. La variable `coul_ter` est une matrice à 1 ligne et 3 colonnes. Chaque colonne est codée sur 8 bit (car pouvant prendre ses valeurs entre 0 et 255), au total, la variable utilise 24 bits.

5.
```
def maillot(image) :
    return [coul(image), [255, 255, 255]]
```

Remarque : on peut s'interroger sur la pertinence d'une telle fonction... Et dans tous les cas critiquer son nom qui ne reflète absolument pas ce qu'elle renvoie.

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, nbB_colonneB - bordure) :
            Bij = mat[i-bordure  : i+bordure+1, j-bordure  : j+bordure+1]
            C[i, j] = np.sum(np.dot(Bij, filtreA))
    return C
```

Remarques :

- il faut bien prendre garde à ne pas itérer sur les cases en «bordure» de l'image ;
- il est discutable de forcer l'écriture de `Bij` en une seule ligne ;
- le sujet en haut de la page 5 se contredit dans la même phrase «on calcule le produit de B_{ij} par A (multiplication classique de $A * B_{ij}$)» : faut-il faire $B_{ij}A$ comme indiqué par le texte ou AB_{ij} comme indiqué par l'expression entre parenthèses ? (nous rappelons aux étudiants que le produit matriciel n'est pas commutatif et qu'ici aucune hypothèse explicitement formulée sur A ne permet de le supposer)

7. Il s'agissait simplement d'appeler la fonction précédente pour chacune des «couches» de l'image.

```
def filtrer(filtreA, matB) :
    C = matB.copy()
    for i in range(3) :
        C[ :,  :, i] = filtre1(filtreA, matB[ :,  :, i]
    return C
```

8.
```
def matriceFlouGaussien(taille, sigma) :
    mat = zeros([taille, taille])
    taille = taille // 2
    for x in range(-taille, taille + 1) :
        for y in range(-taille, taille + 1) :
            mat[taille + x][taille + y] = 1/(2*sqrt(sigma)) * exp(-(x**2 + y**2)/(2*sigma**
    return mat / np.sum(mat)
```

9. Il s'agit ici de simplement composer deux fonctions précédemment écrites.

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

10. La question est floue vu qu'on ne sait pas comment est représentée cette matrice `resultat`. Nous proposons donc plusieurs manières de faire (dans tous les cas, il s'agit d'une question élémentaire de programmation qu'il faut savoir faire) :

```
# si resultat est une matrice ou un tableau numpy :
x = resultat[ :, 0]

# sinon (par exemple si on utilise une liste de listes), on a plusieurs possibilités :
x = []
for ligne in resultat :
    x.append(ligne[0])

# ou bien :
x = []
for i in range(len(resultat)) :
    x.append(resultat[i][0])

# ou la même chose avec une compréhension de liste :
x = [ligne[0] for ligne in resultat]
# ou :
x = [resultat[i][0] for i in range(len(resultat))]
```

Pour `y_plaR`, c'est pareil en changeant les `0` par des `3`.

11. La question ne paraît pas limpide. L'humble correcteur comprend que pour `x = [1, 2], y_plaR = [3, 2]` il faut générer :

- pour `barre` : `[0., 0.1, ..., 2.9, 3.0, 0., 0.1, ..., 1.9, 2]` ;
- pour `abscisse` : `[1, 1, ..., 1, 2, 2, ..., 2]`

```
barre = []
abscisse = []
for i in range(len(x)) :
    for j in range(y_plaR[i] * 10 + 1) :
        abscisse.append(x[i])
        barre.append(j / 10)
```

Remarque : il existe des fonctionnalités de `Matplotlib` pour afficher des histogrammes... La technique proposée relève du bricolage.

12. Algorithme classique... On se demande bien pourquoi le sujet a besoin de 5 lignes pour introduire cette question.

```
def minMaxMoyenne(valeurs) :
    mini = valeurs[0]
    maxi = valeurs[0]
    somme = 0
    for valeur in valeurs :
        if valeur < mini :
            mini = valeur
        if valeur > maxi :
            maxi = valeur
        somme += valeur
    return (mini, maxi, somme/len(valeurs))
```

```
# l'énoncé ne donnant aucune restriction, on peut aussi faire :
def minMaxMoyenne(valeurs) :
    return (min(valeurs), max(valeurs), sum(valeurs)/len(valeurs))
```

13. Requête classique à savoir faire les yeux fermés !

```
SELECT Nom
FROM Joueurs
WHERE VMA > 13 AND Age > 23
```

14.

```
-- Requête 1 :
SELECT Clubs.Nom
FROM Clubs
JOIN Joueurs ON Joueurs.Id_Club = Clubs.Id_Club
WHERE Salaire > 30000
```

```
-- Requête 2 :
SELECT COUNT(*)
FROM Joueurs
WHERE Id_Club = 31 AND Poste = 'Talonneur'
```

Pour la seconde requête, nous aurions pu effectuer également un `JOIN` sur la table `Clubs` mais le sujet donne l'identifiant du Stade Toulousain.

15. Il ne faut pas oublier que le budget des clubs est donné en millions d'euros et celui des joueurs en euros (choix de conception douteux, probablement destiné à piéger les candidats ; il est fortement déconseillé de mélanger différentes unités dans une vraie base de donnée).
La formule est donc : $\frac{\sum Salaire}{Budget \cdot 10^6} \times 100 = \frac{\sum Salaire}{Budget \cdot 10^4}$.

```
SELECT SUM(Salaire) / (Budget * 10000)
FROM Joueurs
JOIN Clubs ON Joueurs.Id_Club = Clubs.Id_Club
GROUP BY Clubs.Id_Club
```

Remarque : certains moteurs SQL forcent de grouper également par `Budget` car celui-ci est utilisé dans le `SELECT`.

16. Question quasiment hors programme étant donné que la connaissance de la bibliothèque `sqlite3` n'est pas au programme. On peut deviner que le type de `monequipe` est `list` en regardant le code qui suit la question, mais nous ne pouvons dans aucun cas en être certain (on peut définir l'opérateur d'indexation «`l[i]`» sur n'importe quel type d'objet en Python).

17. Erreur ligne 7 : `for i in range(len(liste)):` (on pourrait aussi soustraire 1 à la longueur de la liste car la dernière itération ne fait rien, mais est-ce vraiment une «erreur» ?)
Erreur ligne 10 : remplacer le `>` par un `<`.

18. Dans tous les cas, la complexité est en $O(n^2)$ comparaisons. En effet, si `n` est la longueur de la liste :

- à la première itération de la boucle ligne 7, on effectue `n - 1` comparaisons (le `j` de la boucle ligne 9 varie entre `1` et `n -1`) ;
- à la seconde itération de la boucle ligne 7, on effectue `n - 2` comparaisons ;
- $\vdots$
- à la `n - 1`-ème itération de la boucle ligne 7, on effectue `1` comparaisons.

Au final, on effectue $1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} = O(n^2)$ comparaisons. C'est clairement peu efficace car il existe des tris (comme le tri fusion) qui admettent une complexité en $O(n \ln(n))$.

19. Dans ce genre d'exercice, il faut éviter de traduire littéralement le code et tenter prendre un peu de recul sur celui-ci (quand cela est possible).
L.18 : Défini une fonction de tri : la liste `L` sera triée selon le critère `val` [NdC : pourquoi un tel nom ??] entre les indices `i` et `j`. On reconnaît dans le reste du code une implémentation du tri rapide.
L.20 : On vérifie que la sous-liste à trier n'est pas vide.
L.22 : On partitionne la liste et on récupère l'indice du pivot.

L.24 : On trie récursivement la sous-liste à gauche du pivot.
L.26 : On trie récursivement la sous-liste à droite du pivot.
L.28 : Euh... on renvoie la liste triée (ce qui est maladroit car la liste est modifiée en place).
Pour la modification demandée, utiliser une variable globale est maladroit : il faudrait penser à la réinitialiser entre chaque appel récursif. Suivons l'énoncé qui nous demande de faire renvoyer le nombre d'appels récursifs effectués par la fonction

```
def tri_2(L, val, i, j) :
    nb_appels = 0
    if i < j :
        k = segmente(L, val, i, j)
        nb_appels += tri_2(L, val, i, j) + 1
        nb_appels += tri_2(L, val, i, j) + 1
    return nb_appels
```

20. `tri_2(monequipe, 3, 0, len(monequipe) - 1)`

21. $74{,}25_{10} = 1001010{,}01_2$

22.

0	1	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

23. Chaque flottant est représenté sur 4 octets (un octet est constitué de 8 bits), il y a trois caractéristiques à prendre en compte. L'espace occupé est donc $3000 \times 4 \times 3o = 36ko$, ce qui est relativement faible pour les capacités de mémoire d'un ordinateur actuel.

Avantage/inconvénient du format simple précision : il prend moins de place que le format double mais est moins précis (l'auteur du sujet attend-il autre chose que cette lapalissade ?)

24.

```
def liste_temps(pas, tmax) :
    t = 0
    res = []
    while t <= t_max :
        res.append(t)
        t += pas
    return res
```

25. Vos cours de physique/maths vous donnent $v(t) = K_c U_0(1 - \mathrm{e}^{-t/\tau})$.

```
def vitesse(k, tau, u, temps) :
    res = 0
    for t in temps :
        res.append(k * u * (1 - exp(-t/tau)))
    return res
```

26.

```
def ordre1_euler(Kc, tau, U0, temps) :
    s = 0
    res = [0]
    for i in range(1, len(temps)) :
        dt = temps[i] - temps[i-1]
        s += dt * k * U0 / tau
        res.append(s)
    return res
```

27. et **28.**. L'énoncé ne précise pas les valeurs à prendre pour les différentes constantes... Laissons-les donc indéterminées.

```
import matplotlib.pyplot as plt
for p in [0.2, 0.4, 0.6] :
    tps = liste_temps(tmax, p)
    y = ordre1_euler(Kc, tau, U0, tps)
    plot(tps, x)
```

«Super, maintenant je sais programmer une application de Rugby !»
Étudiant sortant de l'épreuve d'informatique CCP TSI 2017.