

2025 métropole jour 2

Exercice 1

Partie A

1. [0.25 point] 010
2. [0.25 point] espion
3. [0.25 point] Un parcours en largeur.

Partie B

4. [0.5 point] La somme à gauche est 11 et la somme à droite aussi. La répartition est donc parfaite.
5. [0.5 point] La hauteur est 5. C'est le nombre maximal de bits utilisés dans un codage.
6. [0.75 point] Comptons les bits utilisés pour Shannon-Fano :

je_pense,_donc_je_suis
4234243243554434233443

Cela fait un total de 75 bits.

Or pour un codage en ASCII on aurait $22 \times 8 = 176$ bits.

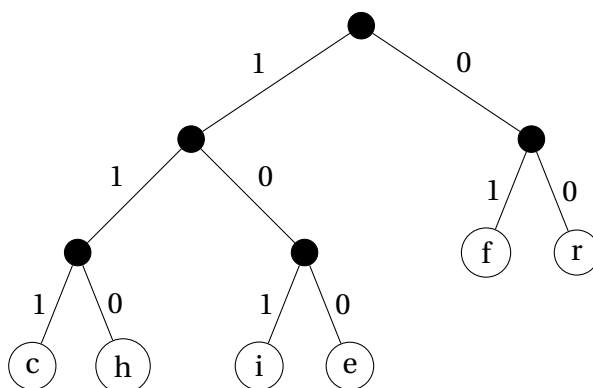
On utilise donc ici moins de deux fois moins de bits avec le codage de Shannon-Fano.

7. [0.75 point]

On trie les lettres par nombre d'occurrences croissant :

| symbole | c | h | i | e | f | r |
|----------------------|---|---|---|---|---|---|
| nombre d'occurrences | 1 | 1 | 1 | 1 | 2 | 2 |

Ensuite, on sépare en sous-groupes récursivement :



Partie C

8. [0.5 point]

```
8    dico[symbole] = dico[symbole] + 1

10   dico[symbole] = 1
```

9. [0.5 point]

```
def somme_occ(tab):
    s = 0
    for symbole, occ in tab:
        s = s + occ
    return s
```

10. [0.5 point]

```
9    return "1" + shannon(symbole, t1)

11   return "0" + shannon(symbole, t2)
```

11. [0.5 point] Le cas d'arrêt est `len(tab) == 1`. Or la taille du tableau est une suite strictement décroissante car à chaque séparation aucune des deux parties ne peut être vide. Donc on arrivera nécessairement à `len(tab) == 1`.

12. [0.75 point]

```
def encode_shannon(texte):
    dico = creer_dico_occ(texte)
    tab = creer_tab_trie(dico)
    s = ""
    for c in texte:
        s = s + shannon(c, tab)
    return s
```

Exercice 2

1. [0.5 point] Plusieurs adhérents peuvent avoir le même nom. Or une clé primaire doit être unique.

2. [0.5 point] Elle affiche le nom des jeux avec leur éditeur par ordre alphabétique du nom du jeu.

3. [0.5 point]

```
SELECT nomJeu
FROM emprunt
WHERE dateRendu = NULL;
```

4. [1 point]

```

SELECT nom, prenom
FROM adherent
JOIN emprunt
ON adherent.idAdherent = emprunt.idAdherent
WHERE nomJeu = 'Catan';

```

5. [0.5 point]

```

UPDATE emprunt
SET dateRendu = '2025-06-3'
WHERE idEmprunt = 1538;

```

6. [0.5 point]

```

SELECT nomJeu, categorie
FROM jeu
WHERE anneeSortie >= 2010 AND ageMinimum < 10;

```

7. [0.5 point]

On aura deux clés étrangères :

- # nom qui fait référence à nom de evenement
- # idAdherent qui fait référence à idAdherent de adherent

8. [0.75 point]

```

dict_emprunts = {}
for jeu in liste:
    if jeu in dict_emprunts:
        dict_emprunts[jeu] += 1
    else:
        dict_emprunts[jeu] = 1

```

9. [1.25 point]

```

def liste_max(dico):
    """ Renvoie une liste de jeux avec le plus grand nombre d'emprunts """
    maxi = 0
    liste = []
    for jeu, nb in dico.items():
        if nb > maxi:
            maxi = nb
            liste = [jeu]
        elif nb == maxi:
            liste.append(jeu)
    return liste

```

```

podium = []
for _ in range(3):
    premiers = liste_max(dict_emprunts)
    # On enlève les premiers pour avoir les suivants
    # au prochain passage
    for jeu in premiers:
        del(dict_emprunts[jeu])
    podium.append(premiers)

```

Exercice 3

Partie A

1. [0.5 point]

```
    11 (L)   8 (I)   1 (B)   17 (R)   4 (E)
+   4 (E)   24 (Y)  16 (Q)   12 (M)  19 (T)
=   15      32      17       29      20
=  15 (P)   6 (G)   17 (R)   3 (D)   23 (X)
```

Le message chiffré est donc PGRDX.

2. [0.5 point]

```
def indice(L, element):
    return L.index(element)
```

Ou alors :

```
def indice(L, element):
    for i in range(len(L)):
        if L[i] == element:
            return i
```

3. [0.5 point]

```
def lettres_vers_indices(texte):
    indices = []
    for c in texte:
        indices.append(indice(alphabet, c))
    return indices
```

4. [0.75 point]

```
for k in range(n):
    ind = indices_msg[k] + indices_cle[k] # ***
    if ind >= 26:
        ind = ind - 26 # ***
    indices_msg_chiffre.append(ind)
msg_chiffre = indices_vers_lettres(indices_msg_chiffre) # ***
return msg_chiffre
```

5. [0.5 point] On obtiendra une assertion error `"impossible"` car la taille de la clé est inférieure à celle du message.

6. [0.5 point]

$$\begin{array}{r}
6 \text{ (G)} \quad 12 \text{ (M)} \quad 4 \text{ (E)} \quad 3 \text{ (D)} \quad 7 \text{ (H)} \\
- \quad 5 \text{ (F)} \quad 21 \text{ (V)} \quad 4 \text{ (E)} \quad 8 \text{ (I)} \quad 19 \text{ (T)} \\
= \quad 1 \quad -9 \quad 0 \quad -5 \quad -12 \\
= \quad 1 \text{ (B)} \quad 17 \text{ (R)} \quad 0 \text{ (A)} \quad 21 \text{ (V)} \quad 14 \text{ (O)}
\end{array}$$

Le message est BRAVO.

7. [0.25 point] On retranche au rang de chaque lettre du message le rang de la lettre correspondante du masque. Pour les valeurs négatives on ajoute 26 (modulo 26).

8. [0.5 point]

```

indices_msg_dechiffre = []
for k in range(n):
    ind = indices_msg[k] - indices_cle[k]
    if ind < 0:
        ind = ind + 26
    indices_msg_dechiffre.append(ind)
msg_dechiffre = indices_vers_lettres(indices_msg_dechiffre)
return msg_dechiffre

```

Partie B

9. [0.5 point] Un algorithme de chiffrement symétrique utilise la même clé pour chiffrer et déchiffrer. Un algorithme de chiffrement asymétrique utilise une clé publique pour chiffrer et une clé privée pour déchiffrer.

10. [0.25 point] Il doit utiliser sa clé privée.

11. [0.5 point] Une autre personne peut connaître la clé publique de Bob comme elle est publique. Rien n'empêche alors cette personne de se faire passer pour Alice.

12. [0.5 point] Le serveur et le client échangent une clé de chiffrement symétrique en utilisant un chiffrement asymétrique. Une fois la clé en possession de chacun, ils chiffrent tout leurs messages avec la clé symétrique.

13. [0.5 point] Car le chiffrement symétrique est plus rapide et moins gourmand en ressources.

Partie C

14. [0.25 point] Le ping n'a pas fonctionné, il s'est trompé dans l'adresse. Il fallait écrire :

```
ping 192.168.110.115
```

15. [0.25 point] 255.255.255.224

16. [0.25 point] Il ya 5 bits disponibles donc $2^5 = 32$ adresses.

17. [0.25 point] $134 = 128 + 4 + 2$ donc 10000110 en binaire.

18. [0.75 point] Il faut savoir si Zoé est sur le même sous-réseau que Bob ou Marc. Calculons le sous-réseau de Zoé :

```

1 1 1 0 0 0 0 0 (224)
ET 1 0 0 0 0 1 1 0 (134)

```

1 0 0 0 0 0 0 0 (128)

Alice est sur le sous-réseau 192.168.110.128 / 27

Calculons le sous-réseau de Marc :

$153_{10} = 10011001_2$

1 1 1 0 0 0 0 0 (224)

ET 1 0 0 1 1 0 0 1 (153)

1 0 0 0 0 0 0 0 (128)

Marc est sur le sous-réseau 192.168.110.128 / 27

Zoé et Marc sont donc sur le même sous-réseau (contrairement à Bob) c'est donc la commande 2 pour laquelle le ping a fonctionné.