2025 métropole jour 1

Exercice 1

Partie A

1. Le numéro de série n'est pas unique. Deux guitares de marques différentes peuvent avoir le même numéro de série comme la 4 et la 8 de l'extrait.

2.

marque	modele
Gibson	Les Paul Goldtop
Fender	Stratocaster

3.
```sql
SELECT annee FROM inventaire WHERE modele = 'Les Paul Standard';
```

4.

```sql
SELECT modele
FROM inventaire
WHERE marque = 'Gibson'
ORDER BY annee;
```

5.
```sql
UPDATE inventaire SET annee = 1957 WHERE id = 1;
```

Partie B

6. `guitare` doit être crée après `modele` car une clé étrangère lui fait référence. `modele` doit être crée après `marque` car une clé étrangère lui fait référence. Donc l'ordre de création doit être `marque`, `modele` puis `guitare`

7.

```sql
SELECT num_ser, annee
FROM guitare
JOIN modele
ON guitare.id_modele = modele.id
WHERE nom = 'Les Paul Standard';
```

8.
```sql
DELETE FROM guitare WHERE id = 3;
```

9.

```sql
INSERT INTO marque VALUES (3, 'BC Rich')
INSERT INTO modele VALUES (5, 'Mockingbird', 3)
INSERT INTO guitare VALUES (9, 5, 1992, '92R', 5000)
```

10.

```sql
SELECT SUM(prix)
FROM guitare
JOIN modele
ON guitare.id_modele = modele.id
WHERE nom = 'Stratocaster';
```

Exercice 2

1.

```
tache1 = Tache(1, "Répondre aux e-mails", 45)
tache2 = Tache(2, "Ranger ma chambre", 60)
```

2.

```
def avancer(self, n):
    self.duree_restante = self.duree_restante - n
```

3.

```
def est_terminee(self):
    return self.duree_restante <= 0
```

4.

```
[début] (<t3>, 4) (<t7>, 4) (<t1>, 3) (<t2>, 3) (<t6>, 2) (<t4>, 1)(<t5>, 1) [fin]
```

5.

```
<t3>
[début] (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1)[fin]
```

6.

```
4
[début] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1)[fin]
```

7.

```
def ajouter_file_prio(f, t, p):
    f_aux = File()
    while not f.est_vide() and f.examiner()[1] >= p:
        f_aux.enfiler(f.defiler())
    f_aux.enfiler((t, p))
    while not f.est_vide():
        f_aux.enfiler(f.defiler())
    while not f_aux.est_vide():
        f.enfiler(f_aux.defiler())
```

8. On defiler deux fois une file de taille m. Le coût est donc linéaire : $O(m)$

9. `3 7 3 3 3 1 2 1 2 2 6 6 6 4 5 4 5`

10.

```
def planning(f):
    tab = []
    while not f.est_vide():
        t, p = f.depiler()
        # On peut aussi ne pas mettre le __repr__()
        tab.append(t.__repr__())
        t.avancer(25)
        if not t.est_terminee():
            ajouter_file_prio(f, t, p)
    return tab
```

Exercice 3

Partie A

1.
Les deux valides sont (a) et (b) car (d) est sur un sous-réseau différent et (c) n'existe pas car 261 > 255.

2. 192.168.20.255

3. Les valeurs vont de 0 à 255, il y en a donc 256. Il faut retrancher l'adresse du réseau (0), l'adresse de broadcast (255), l'adresse du reouteur et les trois autres. Il reste donc 250 possibilités.

4. 8 adresses nécessitent 3 bits. Le masque peut donc utiliser $32 - 3 = 29$ bits.

Partie B

5.

Réseau destination	Interface de sortie	Prochain routeur	Nombre de sauts
192.168.30.0	172.16.4.1	172.16.4.2	1
172.16.1.0	172.16.3.1	172.6.3.2	1

6. Celui qui est attendu semble être `192.168.10.0` :

Réseau destination	Interface de sortie	Prochain routeur	Nombre de sauts
192.168.10.0	172.16.4.1	172.16.4.2	2

Mais il y a aussi le `172.16.0.0` :

Réseau destination	Interface de sortie	Prochain routeur	Nombre de sauts
172.16.0.0	172.16.3.1	172.16.3.2	1

7.

Réseau destination	Interface de sortie	Prochain routeur	Nombre de sauts
autre	172.16.3.1	172.16.3.2	

Partie C

8.
- Fast Ethernet : 10
- Fibre optique : 1

9. Le chemin est 1-2-3-4 avec un coût de 21.

Partie D

10. `'11000000.10101000.00010100.00001100'`

11. Quand les deux adresses sont identiques.

12.

```
def precede(ip_1, ip_2):
    for i in range(35):
        if ip_1[i] < ip_2[i]:
            return True
        elif ip_1[i] > ip_2[i]:
            return False
    return False
```

13.
- attribut : `adresse_ip`
- méthode : `est_vide`

14. `return self.adresse_ip == ''`

15. L'accès aux éléments d'un ABR est rapide, il se fait en $O(log(n))$. Alors que dans un tableau la recherche est linéaire en $O(n)$.

16.

```
def modifie(self, adresse_ip, interface, passerelle, cout):
    if self.est_vide():
        self.gauche = Abr('','','',0)
        self.droite = Abr('','','',0)
    self.adresse_ip = adresse_ip
    self.interface = interface
    self.passerelle = passerelle
    self.cout = cout
```

17.

```
elif precede(ip_bin(adresse_ip), ip_bin(self.adresse_ip)):
```

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 :

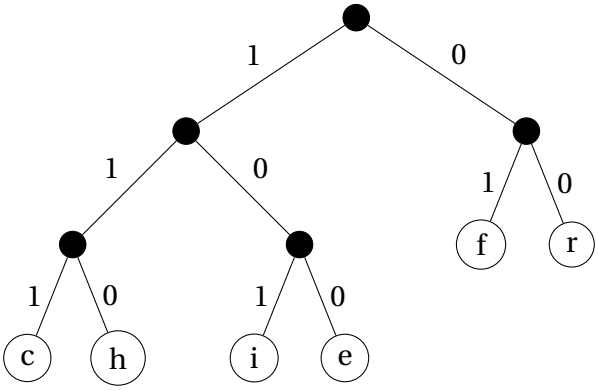
```
je_pense,_donc_je_suis
423424324355443423344 3
```

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 afiche 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]

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

5. [0.5 point]

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

6. [0.5 point]

```sql
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]

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

9. [1.25 point]

```python
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]

	6 (G)	12 (M)	4 (E)	3 (D)	7 (H)
-	5 (F)	21 (V)	4 (E)	8 (I)	19 (T)
=	1	-9	0	-5	-12
=	1 (B)	17 (R)	0 (A)	21 (V)	14 (O)

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 assymétrique 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 connaitre 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.

2024 métropole jour 1

Exercice 1

1. Aucun site ne pointe vers *site2*. Il n'a donc pas de prédécesseurs : la liste est vide.

2.

```
s4.predecesseurs = [(s1, 1), (s2, 2)]
s5.predecesseurs = [(s1, 2), (s3, 3), (s4, 6)]
```

3. `5`, c'est le deuxième élément du deuxième tuple.

4. `6`

5.

```
def calculPopularite(self):
    p = 0
    for t in self.predecesseurs:
        p = p + t[1]
    self.popularite = p
    return p
```

6. File.

7. Parcours en largeur.

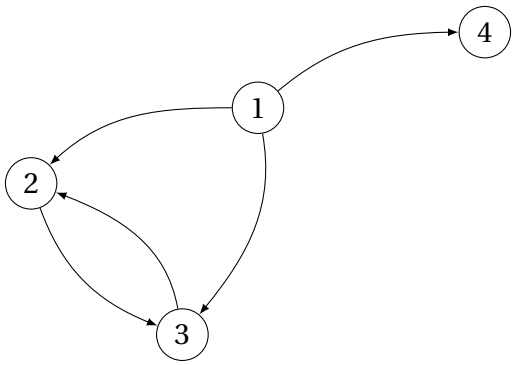
8. `[s1, s3, s5]`

9.

```
maxPopularite = site.popularite
siteLePlusPopulaire = site
```

10. `site3`

11. Ce code devrait fonctionner, mais il est possible de « rater » des sites s'ils ne sont pas pointés par le premier groupe de sites. Par exemple :



si on commence par 2, on ne visitera jamais 4.

Exercice 2

Partie A

1.
- Persistance des données ;
- gestion des accès concurrents ;
- efficacité des traitements ;
- sécurisation des accès.

2. B - E - A

3.
- C - I - G - F - D - A : 2.3
- C - I - H - F - D - A : 2.3

Partie B

4. C'est un entier unique propre à chaque client.

5. Une clé étrangère est une référence vers une clé primaire d'une autre table.
Pour `reservations` :
- `id_client (clients : id_client)`
- `nom_croisiere (croisieres : nom)`

Pour croisieres :
- `escale_1 villes : nom)`
- `escale_2 villes : nom)`
- `escale_3 villes : nom)`
- `escale_4 villes : nom)`

6. Il n'est pas possible d'ajouter une valeur dans une clé étrangère si cette valeur n'existe pas dans la clé primaire référencée.
Il faudrait préalablement ajouter une ligne pour chaque escale dans la table `villes`.

Partie C

7. Il trouve l'`id_client` du monsieur puis cherche la réservation associée.

8.

```sql
SELECT id_reservation
FROM reservations
JOIN clients
ON reservations.id_client = clients.id_client
WHERE nom = 'Barc' AND prenom = 'Jean'
AND date_naissance = '1972/06/29' AND pays = 'Allemagne';
```

9.

```sql
UPDATE reservations
SET nom_croisiere = 'Croisière Puerto'
WHERE id_reservation = 20456;
```

10.

```sql
SELECT nom, prenom, date_naissance
FROM clients
JOIN reservations
ON clients.id_client = reservations.id_client
WHERE nom_croisiere = 'Croisière Piano' OR nom_croisiere = 'Croisière Puerto';
```

Exercice 3

Partie A

1. `chien40 = Chien(40, "Duke", "wheel dog", 10)`

2.

```
def changer_role(self, nouveau_role):
    self.role = nouveau_role
```

3. `chien40.changer_role('leader')`

Partie B

4.

```
def retirer_chien(self, numero):
    liste = []
    for chien in self.liste_chiens:
        if chien.id_chien != numero:
            liste.append(chien)
    self.liste_chiens = liste
```

5. `eq11.retirer_chien(46)`

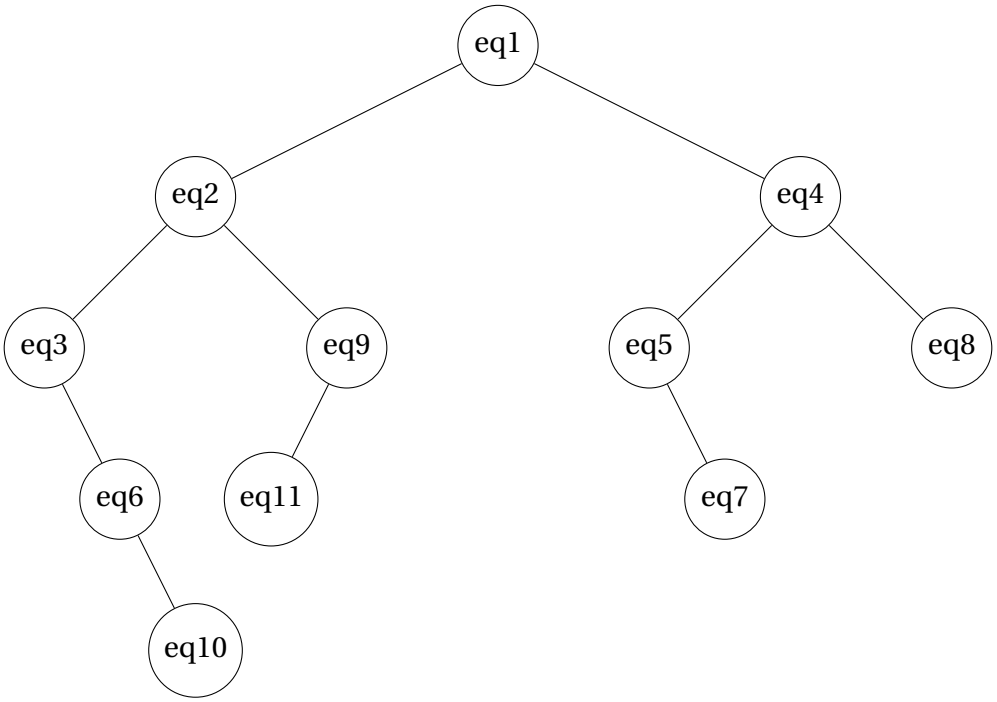
6. $4 + 36/60 = 4.6$

7.

```
def temps_course(equipe):
    temps = 0
    for t in equipe.liste_temps:
        temps = temps + convert(t)
    return temps
```

Partie C

8.



9. Le parcours infixe.

10. Elle s'appelle elle-même à la ligne 2.

11.

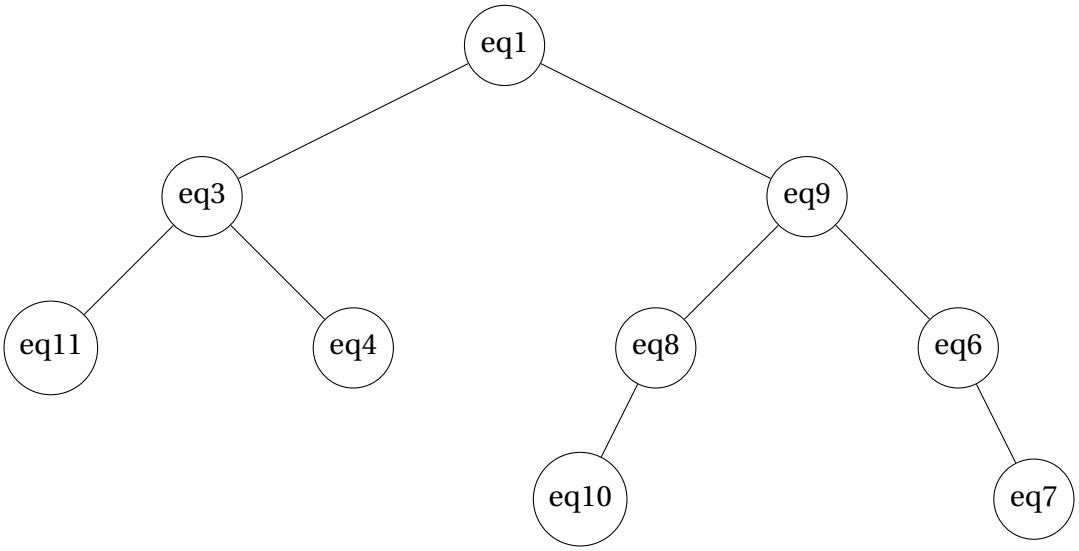
```
def inserer(arb, eq):
    """ Insertion d'une équipe à sa place dans un ABR contenant
        au moins un noeud."""
    if convert(eq.temps_etape) < convert(arb.racine.temps_etape):
        if arb.gauche is None:
            arb.gauche = Noeud(eq)
        else:
            inserer(arb.gauche, eq)
    else:
        if arb.droit is None:
            arb.droit = Noeud(eq)
        else:
            inserer(arb.droit, eq)
```

12.

```
def est_gagnante(arbre):
    if arbre.gauche == None:
        return arbre.racine.nom_equipe
    else:
        return est_gagnante(arbre.gauche)
```

Partie D

13.



14.

```
def rechercher(arbre, equipe):
    if arbre == None:
        return False
    elif arbre.racine = equipe:
        return True
    elif convert(equipe.temps_etape) < convert(arbre.racine.temps_etape):
        return recherher(arbre.gauche, equipe)
    else:
        return rechercher(arbre.droit, equipe)
```

2024 métropole jour 2

Exercice 1

Partie A

1. Non, car les valeurs de ce champ ne sont pas uniques : un artiste peut avoir plusieurs albums et donc apparaître plusieurs fois dans la table `CD`.

2.

Nom_artiste
Nightwish
The Rasmus

3.

Annee
1986
2001
1986

4.
```
UPDATE CD SET Annee = 2000 WHERE id_album = 4;
```

5.

```
SELECT Titre_album
FROM CD
JOIN Artiste ON CD.Nom_artiste = Artiste.Nom_artiste
JOIN Rangement ON CD.id_album = Rangement.id_album
WHERE Style = 'Metal' AND Numero_etagere = 1;
```

6. Il doit d'abord supprimer la ligne avec `id_album = 5` de la table `Rangement` pour supprimer la clé étrangère faisant référence à la clé primaire de la table `CD`.
Ensuite il peut supprimer la ligne avec `id_album = 5` de la table `CD` pour supprimer la clé étrangère faisant référence à la clé primaire de la table `Artiste`.
Enfin il peut supprimer la ligne avec `Nom_artiste = 'The Rasmus'` de la table `Artiste`.
La requête de la suppression de l'album est :

```
DELETE FROM CD WHERE id_album = 5;
```

Partie B

7. Un algorithme de chiffrement symétrique utilise la même clé pour chiffrer et pour déchiffrer un message.

8. Un algorithme de chiffrement asymétrique utilise la clé publique du destinataire pour chiffrer le message et le destinataire utiliser sa clé privée pour le déchiffrer.

9. Le serveur doit chiffrer la clé C avec la clé publique de Bob. Il lui envoie alors la clé C chiffrée. Puis Bob déchiffre cette clé avec sa clé privée.

Exercice 2

Partie A

1.

```
class Marchandise:
    def __init__(self, p: int, v: int) -> 'Marchandise':
        assert v > 0
        self.prix = p
        self.volume = v
```

2. `m1 = Marchandise(20, 7)`

3.

```
def ratio(self) -> float:
    return self.prix / self.volume
```

4.

```
def prixListe(tab: list) -> int:
    somme = 0
    for m in tab:
        somme = somme + m.prix
    return somme
```

Partie B

5.

- m1 -> prix : 40, volume : 20
- m2 -> prix : 210, volume : 70
- m3 -> prix : 160, volume : 40
- m4 -> prix : 50, volume : 50
- m1, m2 -> prix : 250, volume : 90
- m1, m3 -> prix : 200, volume : 60
- m1, m4 -> prix : 90, volume : 70
- m3, m4 -> prix : 210, volume : 90

La meilleure combinaison est m1, m2 avec 250 €.

6. glouton

7.

```
def tri(tab: list) -> None:
    n = len(tab)
    for i in range(1, n):
        marchandise = tab[i]
        j = i-1
        while j >= 0 and marchandise.ratio() > tab[j].ratio() :
            tab[j+1] = tab[j]
            j = j-1
        tab[j+1] = marchandise
```

8. C'est un tri par insertion, son coût est quadratique.

9.

```
def charge(tab: list, volume: int) -> list:
    tri(tab)
    chargement = []
    n = len(tab)
    for i in range (n):
        if tab[i].volume <= volume:
            chargement.append(tab[i])
            volume = volume - tab[i].volume
    return chargement
```

Partie C

10.
[Attention, `n` n'est pas défini, il faut donc utiliser `len(tab)`. Et il y a une interversion entre l'option 1 et l'option 2 de l'énoncé.]

```
def chargeOptimale(tab: list, v_restant: int, i: int) -> list:
    if i >= len(tab):
        return []
    else:
        if tab[i].volume > v_restant:
            return chargeOptimale(tab, v_restant, i+1)
        else:
            option1 = chargeOptimale(tab, v_restant, i+1)
            option2 = [tab[i]] + chargeOptimale(tab, v_restant - tab[i].volume, i+1)
            if prixListe(option1) > prixListe(option2):
                return option1
            else:
                return option2
```

Exercice 3

Partie A

1.

— nom : string
— denivele : int
— longueur : float
— couleur : string
— ouverte : booléen

2.

```
def set_couleur(self):
    if self.denivele >= 100:
        self.couleur = 'noire'
    elif self.denivele >= 70:
        self.couleur = 'rouge'
    elif self.denivele >= 40:
        self.couleur = 'bleue'
    else:
        self.couleur = 'verte'
```

3. D

4.

```
for piste in lievre_blanc.get_pistes():
    if piste.get_couleur() == 'verte':
        piste.ouverte = False
```

5.
[Légère incohérence entre l'ordre des paramètres de l'énoncé et celui de l'exemple.]

```
def pistes_de_couleur(couleur, lst):
    liste_noms = []
    for piste in lst:
        if piste.get_couleur() == couleur:
            liste_noms.append(piste.get_nom())
    return liste_noms
```

6.

```
def semi_marathon(L):
    distance = 0
    liste_pistes = lievre_blanc.get_pistes()
    for nom in L:
        for piste in liste_pistes:
            if piste.get_nom() == nom:
                distance = distance + piste.get_longueur()
    return distance > 21.1
```

Partie B

7. `domaine['E']['F']`

8.

```
def voisins(G, s):
    liste_voisins = []
    for v in G[s].keys():
        liste_voisins.append(v)
    return liste_voisins
```

9.

```
def longueur_chemin(G, chemin):
    precedent = chemin[0]
    longueur = 0
    for i in range(1, len(chemin)):
        longueur = longueur + G[precedent][chemin[i]]
        precedent = chemin[i]
    return longueur
```

10. La fonction parcours est récursive car elle s'appelle elle-même à la ligne 7.
[La fonction `parcours` est très mal écrite. Si on l'appelle deux fois de suite, il n'y aura pas le même résultat car l'initialisation par défaut d'un objet mutable (liste) n'est faite qu'au premier appel de la fonction. Ainsi, `lst_chemins` ne sera vide qu'au premier appel de la fonction ensuite elle contiendra les chemins déjà trouvés. L'appel récursif à la ligne 7 utilise cette propriété pour fonctionner mais c'est très dangereux! Il est préférable de ne pas utiliser de valeurs par défaut pour les objets mutables pour éviter des comportements étranges. Voilà comment il aurait fallu écrire la fonction pour éviter des bugs :]

```
def parcours(G, depart, chemin, lst_chemins):
    if chemin == []:
        chemin = [depart]
    for sommet in voisins(G, depart):
        if sommet not in chemin:
            lst_chemins.append(chemin + [sommet])
            parcours(G, sommet, chemin + [sommet], lst_chemins)
    return lst_chemins
```

11.
[Erreur dans le sujet : il n'y a pas de doublons dans la liste renvoyée par `parcours` ! Des doublons apparaissent à cause de l'écriture dangereuse de `parcours` et de son appel sans donner des listes vides dans la fonction `parcours_dep_arr`. Le corrigé ci-dessous empêche le mauvais comportement même avec l'écriture risquée de `parcours` (il n'y a donc pas besoin d'éliminer les doublons car il n'y en a pas) :]

```
def parcours_dep_arr(G, depart, arrivee):
    liste = parcours(G, depart, [], []) # Force une nouvelle initialisation
    lst_chemins = []
    for chemin in liste:
        if chemin[-1] == arrivee:
            lst_chemins.append(chemin)
    return lst_chemins
```

[Voilà la fonction attendue qui enlève les doublons créés par l'écriture risquée de `parcours` :]

```python
def parcours_dep_arr(G, depart, arrivee):
    liste = parcours(G, depart)
    lst_chemins = []
    for chemin in liste:
        if chemin[-1] == arrivee:
            if not chemin in lst_chemins: # Inutile en théorie
                lst_chemins.append(chemin)
    return lst_chemins
```

12.

```python
def plus_court(G, depart, arrivee):
    liste_chemins = parcours_dep_arr(G, depart, arrivee)
    chemin_plus_court = liste_chemins[0]
    minimum = longueur_chemin(G, chemin_plus_court)
    for chemin in liste_chemins:
        longueur = longueur_chemin(G, chemin)
        if longueur < minimum:
            minimum = longueur_chemin(G, chemin)
            chemin_plus_court = chemin
    return chemin_plus_court
```

13. Le dénivelé peut jouer un rôle important dans le temps de parcours d'une piste. Il faudrait donc mesurer le temps effectif de parcours des pistes et calculer le temps minimum plutôt que la distance minimum.

2024 Métropole septembre jour 1

Exercice 1

Partie A

1. Ce champ pourrait être de type entier (INT).

2. 5 (c'est le nombre d'agences avec un numéro de téléphone distinct, mais elles ont toutes un téléphone différent...)

3. Il faut qu'aucune agence n'ai le même numéro car une clé primaire doit être unique.

4. `couple_voitures_agences : (#id_agence : INT, #id_voiture : INT)`
La clé primaire est le couple `(id_agence, id_voiture)` et les deux champs sont des clés étrangères.
[On peut aussi créer un champ supplémentaire pour une clé primaire mais c'est inutile]

5.

```
INSERT INTO couple_voitures_agences
VALUES (5, 2);
```

6.

```
UPDATE couple_voitures_agences
SET id_agence = 2
WHERE id_agence = 5 AND id_voiture = 2;
```

7.

```
SELECT type, marque, Agence
FROM Agences
JOIN couple_voitures_agences
ON Agences.id_agence = couple_voitures_agences.id_agence
JOIN  Voitures
ON Voitures.id_voiture = couple_voitures_agences.id_voiture;
```

Partie B

8.
[On ne peut pas créer une telle fonction car il faut l'id_voiture de la voiture et rien ne nous permet de le récupérer.]

```
def insert_voiture(lv, id_agence):
    req = "INSERT INTO Voitures
    (marque, modele, kilometrage, nombre_places, type, carburant)
    VALUES ("+lv[0]+","+lv[1]+","+lv[2]+","+lv[3]+","+lv[4]+","+lv[5]+");"
    # On suppose que l'id_voiture est renvoyé sinon c'est impossible
    id_voiture = execute_requete_insert(req)

    req = "INSERT INTO couple_voitures_agences
    VALUES ("+id_agence+","+id_voiture+");"
    rep = execute_requete_insert(req)
    return rep
```

9. `lv` doit bien posséder les 6 paramètres de la voiture et id_agence doit exister dans la table `Agences`.

Exercice 2

Partie A

1. Il est sur le même réseau que P1 donc `192.168.1.11`.

2. L'adresse du réseau est `192.168.18.0/24`. On enlève généralement les deux adresses (en 0 et 255), il reste donc $256 - 2 = 254$ adresses possibles.

Partie B

3.

```
G  = {  "R1" : ["R2", "R3", "R4", "R6"],
        "R2" : ["R1", "R3", "R5"],
        "R3" : ["R1", "R2", "R5", "R6"],
        "R4" : ["R1", "R6"],
        "R5" : ["R2", "R3", "R6", "R7"],
        "R6" : ["R1", "R3", "R4", "R5", "R7", "R8"],
        "R7" : ["R5", "R6", "R8", "R9"],
        "R8" : ["R6", "R7", "R9"],
        "R9" : ["R7", "R8"]
    }
```

4.

Destination	Suivant	Nombre de sauts
R2	R2	1
R3	R3	1
R4	R4	1
R5	R3	2
R6	R6	1
R7	R6	2
R8	R6	2
R9	R6	3

5.
R1 - R6 - R8 - R9 avec 3 sauts.

6.

```
M  =   [[0, 1, 1, 1, 0, 1, 0, 0, 0],
        [1, 0, 1, 0, 1, 0, 0, 0, 0],
        [1, 1, 0, 0, 1, 1, 0, 0, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 1, 1, 0, 0, 1, 1, 0, 0],
        [1, 0, 1, 1, 1, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 1, 0, 1, 1],
        [0, 0, 0, 0, 0, 1, 1, 0, 1],
        [0, 0, 0, 0, 0, 0, 1, 1, 0]]
```

7.

```
def degre(MATRICE):
    d = []
    for ligne in MATRICE:
        cpt = 0
        for e in ligne:
            cpt = cpt + e
        d.append(cpt)
    return d
```

8. `[4, 3, 4, 2, 4, 6, 4, 3, 2]`

9. Ce graphe possède deux sommets de degré impair et il est connexe. Il admet donc une chaîne eulérienne. Donc le robot pourra parcourir l'ensemble du réseau en empruntant chaque fibre une et une seule fois.

Partie C

10.
On calcule les coût pour chaque type de liaison :

— $100 : c = \frac{10^8}{100 \times 10^6} = 1$
— $50 : c = \frac{10^8}{50 \times 10^6} = 2$
— $10 : c = \frac{10^8}{10 \times 10^6} = 10$

Le chemin sera : R1 - R2 - R5 - R6 - R7 - R9 avec un coût de 6.

Exercice 3

1. Les quatre cases manquantes :

$2 \times 5 + 1 \times 1 = 11$	$11 \times 20^1 = 220$
$3 \times 5 + 0 \times 1 = 15$	$15 \times 20^0 = 15$

2. On doit ajouter les résultats de la question précédente : $3200 + 220 + 15 = 3435$.

3.

```
M = Maya()
M.ajouter([0, 0, 3])
M.ajouter([0, 1, 2])
M.ajouter([0, 3, 1])
```

4.

```
def nbEtages(self) :
    return len(self.nombre)
```

5.

```
def valeurChiffre(L):
    return L[1] + 5 * L[2]
```

6.

```
def MayaToDec(self):
    coeff = 20**(self.nbEtages() - 1)
    ch_Dec = 0
    while not self.estVide() :
        ch_Maya = self.retirer()
        ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
        coeff = coeff // 20
    return ch_Dec
```

7.

```
def decompChiffre(n):
    if n == 0:
        c = 1
    else:
        c = 0
    p = n % 5
    t = n // 5
    return [c, p, t]
```

8.

```
def DecToMaya(n):
    b20 = DecToVige(n)
    M = Maya()
    for e in b20:
        M.ajouter(decompChiffre(e))
    return M
```

9.

```
def multiplie(self):
    self.nombre = [[1, 0, 0]] + self.nombre
```

10.
[Il y a un problème d'indentation dans la fonction, un = devrait être un == et (m1[0] == 1 and m2[0] == 1).]

```
>>> mystere([0, 1, 1], [0, 3, 1], 0)
([0, 4, 2], 0)
>>> mystere([0, 1, 1], [0, 4, 2], 0)
([1, 0, 0], 1)
```

11.

```
def somme(self, maya2):
    r = 0
    res = Maya()
    for i in range(self.nbEtages()):
        s, r = mystere(self.nombre[i], maya2.nombre[i], r)
        res.ajouter(s)
    if r == 1:
        res.ajouter([0, 1, 0])
    return res
```