BACCALAURÉAT BLANC GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MARDI 16 DECEMBRE 2025

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 15 pages numérotées de 1 / 16 à 16 / 16

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur les protocoles réseaux, l'algorithmique et la POO.

Partie A

Un cœur de réseau est constitué d'un maillage dans lequel les routeurs R_1 à R_6 sont reliés par des liaisons physiques dont le débit en Mbps (Méga-bits par seconde) est indiqué sur la figure suivante.

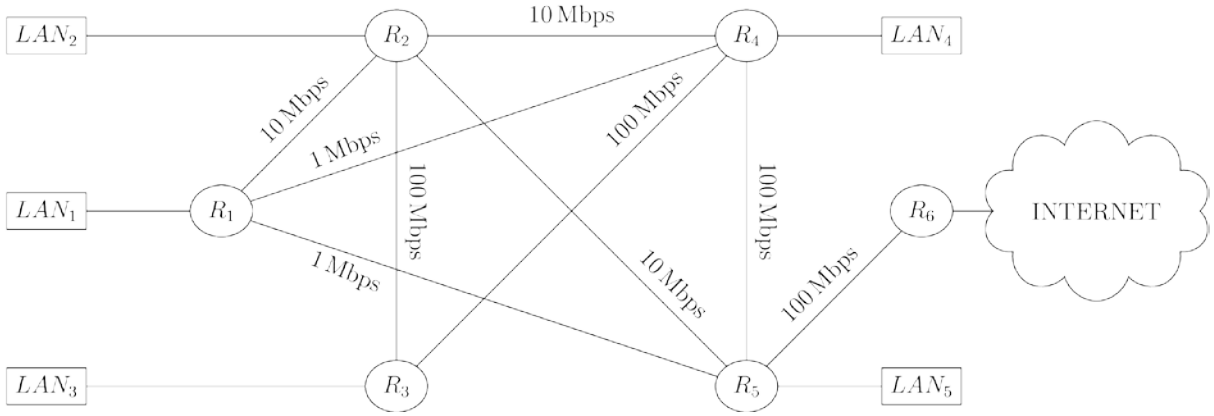

Figure 1. Réseau

Derrière chaque routeur R_1 à R_5, un switch distribue un réseau local LAN_1 à LAN_5.

Les protocoles utilisés sont le protocole RIP et le protocole OSPF, qui minimisent respectivement le nombre de routeurs traversés et la somme des coûts des liaisons physiques empruntées.

1. Recopier et compléter la table de routage de R_1 sachant que le protocole de routage RIP est utilisé.

   La destination est le routeur à atteindre, le prochain saut est le premier routeur traversé et la distance est le nombre de routeurs traversés.

Table de routage R_1		
destination	prochain saut	distance
R_2	R_2	0
R_3	R_2	1
R_4		
R_5		
R_6		

2. Un utilisateur du réseau local LAN_1 interroge, via son navigateur web, un serveur d'Internet. Détailler, suivant le protocole RIP, la route suivie dans le maillage par la requête à destination de ce serveur.

Le coût d'une liaison est donné par la relation coût = $\frac{10^2}{d}$, où d est le débit de la liaison en Mbps.

Exemple : le coût de la liaison entre R_1 et R_2 est $\frac{10^2}{10} = 10$.

Le tableau suivant donne le coût de chacune des liaisons physiques entre R_1 et les autres routeurs du graphe qui lui sont connectés.

Coût des liaisons depuis R_1			
routeur	R_2	R_4	R_5
coût	10	100	100

3. Recopier et compléter la table de routage de R_1 sachant que le protocole de routage OSPF est utilisé.

   La distance est la somme totale des coûts des liaisons physiques empruntées pour atteindre la destination.

Table de routage R_1		
destination	prochain saut	distance
R_2	R_2	10
R_3		
R_4		
R_5		
R_6		

Un utilisateur du réseau local LAN_1 interroge, via son navigateur web, un serveur d'Internet.

4. Donner, selon le protocole OSPF, la route suivie dans le maillage par la requête à destination de ce serveur.

Le protocole de routage OSPF est toujours utilisé et le routeur R_2 tombe en panne.

5. Donner la nouvelle route suivie dans le maillage par une requête partant du réseau LAN_1 à destination d'Internet et en donner la nouvelle valeur de la distance.

Partie B

Le réseau LAN_1 est supposé disposer d'un serveur DHCP (Dynamic Host Control Protocol) gérant l'attribution d'adresses IPv4. Il est rappelé, ici, qu'une adresse IPv4 est une séquence de 4 octets, généralement représentée par les 4 entiers correspondants en écriture décimale, séparés par des points (notation décimale pointée). Un réseau est caractérisé par des machines dont les adresses ont en commun les n mêmes premiers bits. La notation CIDR du réseau a la forme "adresse réseau / n". Appliqué à une adresse du réseau, le masque permet de calculer le préfixe réseau commun. Il est constitué de 4 octets consécutifs, dont (de gauche à droite) les n premiers bits sont égaux à `1` et les suivants à `0`. On obtient l'adresse du réseau en effectuant un `ET` logique, bit à bit, entre chaque octet composant l'adresse d'une machine et l'octet qui lui correspond dans le masque.

Exemple de `ET` :

```
  l'entier 192 s'écrit : 1 1 0 0 0 0 0 0
  l'entier 255 s'écrit : 1 1 1 1 1 1 1 1
-------------------------------------------------------
ET logique, bit à bit  : 1 1 0 0 0 0 0 0 ( soit l'entier 192 )
```

6. Recopier et compléter le tableau suivant correspondant à une machine d'un réseau d'adresse `192.168.1.100` et de masque `255.192.0.0`. Ce tableau détermine l'adresse du réseau (en binaire puis en décimale pointée).

Calcul d'une adresse de réseau				
machine (binaire)	11000000	10101000		
masque (binaire)	11111111	11000000	00000000	00000000
réseau (binaire)	11000000			
réseau (déc. pointée)	`192`			

On obtient le complémentaire d'un octet en échangeant respectivement chacun des 0 et des 1 qui le composent, par un 1 ou un 0.

Par exemple, le complémentaire de `1 1 0 0 1 0 1 0` est `0 0 1 1 0 1 0 1`.

Par un procédé analogue à celui de la question précédente, l'adresse de broadcast d'un réseau est obtenue en effectuant un `OU` logique bit à bit entre chaque octet composant l'adresse réseau et le complémentaire de l'octet correspondant dans l'écriture binaire du masque.

Exemple de OU :

```
          12 s'écrit : 0 0 0 0 1 1 0 0
           9 s'écrit : 0 0 0 0 1 0 0 1
-------------------------------------------------------------
OU logique, bit à bit : 0 0 0 0 1 1 0 1 ( soit l'entier 13 )
```

7. Recopier et compléter le tableau ci-après pour déterminer l'adresse de broadcast du réseau suivant (en binaire puis en décimale pointée).

Calcul d'une adresse de broadcast				
réseau (binaire)	11000000	10000000	00000000	00000000
masque (binaire)	11111111	11000000	00000000	00000000
complément du masque (binaire)				
broadcast (binaire)				
broadcast (déc. pointée)				

Une machine du réseau LAN_1 a reçu du serveur DHCP l'adresse `172.16.1.100`, avec le masque `255.255.0.0`.

Sa passerelle par défaut a pour adresse `172.16.255.254`.

8. En déduire les informations suivantes :
   - l'adresse du réseau LAN_1 (en décimale pointée) ;
   - l'adresse de broadcast (en décimale pointée) ;
   - le nombre total d'adresses pouvant être distribuées par le serveur, en ne tenant pas compte des éventuelles restrictions possiblement posées par l'administrateur du réseau.

On souhaite, désormais, simuler en `Python` le fonctionnement du réseau LAN_1.

On donne, ci-après, les documentations des méthodes `split` et `join`.

- la documentation de la méthode `split` de la classe `str` est la suivante :

```
split(self, séparateur)
  Renvoie la liste des sous-chaines de caractères
délimitées par le séparateur dans l'instance courante de
chaîne de caractères.
  Exemple :
>>> 'ab-pq-rs'.split('-')
['ab', 'pq', 'rs']
```

- la documentation de la méthode `join` de l'objet `str` :

```
join(self, iterable)
Fusionne les chaînes de caractères contenues dans iterable
en le liant par la sous-chaîne depuis laquelle cette
méthode est appelée.
Exemple:
>>>'.'.join(['ab', 'pq', 'rs'])
'ab.pq.rs'
```

On commence par créer la classe `IPv4` suivante.

```
 1 class IPv4(object):
 2     def __init__(self, adresse:str):
 3         """
 4         Constructeur de la classe de calcul sur une
 5         adresse IPv4 dont la notation décimale pointée
 6         est passée en paramètre.
 7         """
 8         self.adresse = adresse
 9
10     def octets(self)->list[int]:
11         """
12         Découpe l'adresse décimale pointée en la liste
13         des 4 entiers correspondants.
14         """
15         return [int(i) for i in self.adresse.split(".")]
```

Pour mémoire, l'opérateur `&` entre deux entiers effectue le `ET` logique bit à bit de la représentation binaire de ces entiers et renvoie l'entier correspondant à la représentation binaire ainsi obtenue.

Exemple :

```
>>> 192 & 255
192
```

9. Recopier sur la copie et compléter les lignes 15 et 16 du code de la méthode `masquer` de la classe `IPv4`, donné ci-dessous. La méthode doit correspondre au docstring.

```
 1      def masquer(self, masque: str)->str:
 2          """
 3          Détermine le préfixe masqué de l'adresse,
 4          le masque (décimal pointé) étant passé en
 5          paramètre.
 6          >>> add  = IPv4('192.168.1.100')
 7          >>> add.masquer('255.192.0.0')
 8          '192.128.0.0'
 9          """
10          tmp = []
11          ip = self.octets()
12          crible = IPv4(masque).octets()
13          for i in range(4):
14              # Opération booléenne :
15              tmp.append(... & ...)
16          return ".".join(...)
```

10. Recopier sur la copie et compléter les lignes 19 et 20 du code de la méthode `adresse_suivante` de la classe `IPv4`, donné ci-dessous. La méthode doit correspondre au docstring.

```
 1      def adresse_suivante(self, adresse_max:str)->str:
 2          """
 3          Détermine l'adresse décimale pointée suivant
 4          immédiatement l'adresse courante, sous réserve
 5          d'existence d'une adresse disponible
 6          >>> add = IPv4('192.168.1.100')
 7          >>> add.adresse_suivante('192.168.1.254')
 8          '192.168.1.101'
 9          >>> add = IPv4('192.168.1.255')
10          >>> add.adresse_suivante('192.168.255.254')
11          '192.168.2.0'
12          """
13          assert self.adresse < adresse_max
14          liste_courante = self.octets()
15          liste_suivante = list()
16          retenue = 1
17          for index in range(4):
18              somme = liste_courante[3 - index] + retenue
19              valeur, retenue = ..., ...
20              liste_suivante = ...
21          return '.'.join(liste_suivante)
```

Exercice 2 (6 points)

Une médiathèque souhaite organiser et gérer sa collection d'albums de différents genres musicaux. On souhaite conserver les informations suivantes sur les albums :

- l'identifiant de l'album (`id`),
- le titre (`titre`),
- le nom de l'artiste ou du groupe (`artiste`),
- le genre musical (`genre`),
- l'année de sortie (`ann_sortie`),
- une note sur 10 attribuée par les utilisateurs (`note`).

Voici un extrait des informations à gérer :

id	titre	artiste	genre	ann_sortie	note
1	Thriller	Michael Jackson	Pop	1982	10
2	Back in Black	AC/DC	Rock	1980	9
3	Deux frères	PNL	Rap	2019	10
4	The Eminem Show	Eminem	Rap	2002	9
5	Nevermind	Nirvana	Grunge	1991	9
6	To Pimp a Butterfly	Kendrick Lamar	Rap	2015	10
7	Trône	Booba	Rap	2017	8
8	Random Access Memories	Daft Punk	Electro	2013	8
9	Wish You Were Here	Pink Floyd	Rock	1975	10

Partie A : Utilisation d'un dictionnaire Python

On considère le dictionnaire suivant :

```
mediatheque = {
    'id': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    'titre': ['Thriller', 'Back in Black', 'Deux frères',
              'The Eminem Show', 'Nevermind', 'To Pimp a Butterfly',
              'Trône', 'Random Access Memories', 'Wish You Were Here
    'artiste': ['Michael Jackson', 'AC/DC', 'PNL',
                'Eminem', 'Nirvana', 'Kendrick Lamar',
                'Booba', 'Daft Punk', 'Pink Floyd'],
    'genre': ['Pop', 'Rock', 'Rap',
              'Rap', 'Grunge', 'Rap',
              'Rap', 'Electro', 'Rock'],
    'ann_sortie': [1982, 1980, 2019, 2002, 1991, 2015, 2017, 2013, 1
    'note': [10, 9, 10, 9, 9, 10, 8, 8, 10]
}
```

1. Quelle est la valeur de la variable x après l'exécution du programme suivant ?

```
x = mediatheque['artiste'][5]
```

2. Compléter la fonction suivante pour qu'elle retourne le genre musical d'un album à partir de son identifiant. Si l'identifiant n'existe pas, la fonction devra renvoyer None.

```
def genre_album(dico, id_album):
    for i in range(len(dico['id'])):
        if dico['id'][i] == ...:
            return dico['genre'][...]
    return ...
```

3. Écrire une fonction artistes_genre qui prend en paramètre un dictionnaire (comme mediatheque) et un genre musical, et qui renvoie une liste contenant les noms des artistes ayant sorti des albums de ce genre.

4. Écrire une fonction meilleure_note qui renvoie une liste des titres des albums ayant obtenu la meilleure note.

Partie B : Programmation orientée objet

On modélise les albums et la médiathèque à l'aide des classes suivantes :

```
class Album:
    def __init__(self, id_album, titre, artiste, genre, ann_sortie,
        self.id = id_album
        self.titre = titre
        self.artiste = artiste
        self.genre = genre
        self.ann_sortie = ann_sortie
        self.note = note

class MediathequeMusicale:
    def __init__(self):
        self.collection = []

    def ajouter_album(self, album):
        self.collection.append(album)

    def trouver_titre(self, id_album):
        for album in self.collection:
            if album.id == id_album:
                return album.titre
        return None
```

5. Ecrire la méthode get_genre de la classe Album pour qu'elle renvoie le genre musical d'un album.

6. Écrire un programme qui crée une instance de la classe MediathequeMusicale, puis ajoute l'album To Pimp a Butterfly, qu'il faudrait créer au préalable, à la collection.

7. Compléter la méthode albums_par_genre de la classe MediathequeMusicale pour qu'elle renvoie une liste des titres des albums d'un genre donné.

```
def albums_par_genre(self, genre):
    ...
```

Partie C : Base de données relationnelle et requêtes SQL

On crée deux tables :

albums

id	titre	id_artiste	genre	ann_sortie	note
1	Thriller	1	Pop	1982	10
2	Back in Black	2	Rock	1980	9
3	Deux frères	3	Rap	2019	10
4	The Eminem Show	4	Rap	2002	9
5	Nevermind	5	Grunge	1991	9
6	To Pimp a Butterfly	6	Rap	2015	10
7	Trône	7	Rap	2017	8
8	Random Access Memories	8	Electro	2013	8
9	Wish You Were Here	9	Rock	1975	10

artistes

id	nom	prenom	annee_naissance
1	Jackson	Michael	1958
2	AC/DC	(Groupe)	1973
3	PNL	(Groupe)	2014
4	Eminem	Marshall	1972
5	Nirvana	(Groupe)	1987
6	Lamar	Kendrick	1987
7	Booba	Élie	1976
8	Daft Punk	(Groupe)	1993
9	Pink Floyd	(Groupe)	1965

8. Écrire une requête SQL qui renvoie les titres des albums du genre Rap publiés après 2000.

9. Expliquer pourquoi il est préférable d'utiliser deux tables (artistes et albums) plutôt qu'une seule.

10. Écrire une requête SQL permettant de modifier la note de l'album **Trône** pour la passer de 8/10 à 9/10.

11. Expliquer le résultat de la requête suivante :

```sql
SELECT nom, prenom
FROM artistes
JOIN albums ON artistes.id = albums.id_artiste
WHERE albums.ann_sortie < 1980;
```

12. Un étudiant décide de développer une application similaire pour gérer des films. En quoi pourrait-il réutiliser les concepts abordés dans cet exercice, la justification et les détails de conception sont valorisés dans votre réponse ?

Exercice 3 (8 points)

Un centre d'appels reçoit des demandes de clients. Chaque demande est stockée dans une structure de **file d'attente**. On souhaite simuler leur fonctionnement.

Chaque demande est un tuple contenant :

- `id` (identifiant unique),
- `client` (nom du client),
- `priorite` (entier entre 1 et 5, 5 = urgent),
- `duree_estimee` (en minutes).

Exemple:

```
demande1 = ("D8", "Raymond", 2, 20)
```

Signifie que la `demande1` a *l'identifiant 8*, qu'elle a été *faite par Raymond*, que sa *priorité est à 2* et qu'elle *prend 20 minutes* à réaliser.

Partie A – Structures de données : piles et files

On considère donc une file d'attente représentée par une liste Python avec les demandes suivantes.
Les demandes sont entrées dans l'ordre de leurs identifiants:

```
file_attente = [
    ("D1", "Alice", 3, 15),
    ("D2", "Bob", 5, 10),
    ("D3", "Chloé", 1, 8),
    ("D4", "David", 4, 20)
]
```

1. Quel élément sera traité en premier dans cette file ? **Justifier**.
2. On souhaite transférer temporairement les demandes dans une **pile**.
   Recopier et compléter le code pour empiler tous les éléments de `file_attente` dans `pile_temp` :
3. Quelle est la différence entre une pile et une file, donner leur type de structure de données?

```
pile_temp = []
for elem in file_attente:
    ...
```

4. Quel élément sera dépilé en premier ? **Expliquer**.

Partie B – Récursivité

On veut calculer **récursivement** la **durée totale estimée des demandes urgentes** c'est à dire uniquement la durée des demande ayant une **priorité supérieure ou égale à 4**.

5. Recopier et compléter le code de la fonction permettant ce calcul:

```
def duree_urgente(file_demande):
    if ... :
        return 0
    id, client, priorite, duree = file_demande[0]
    if priorite >= 4:
        return duree + ...
    else:
        return ...
```

6. Identifier le(s) cas de base et le(s) cas récursif de cette fonction.

7. Réaliser l'arbre d'appels de cette fonction avec la file `file_attente` donnée à la partie précédente.

Partie C – Programmation orientée objet

Pour la suite de l'exercice, chaque demande est un objet `Demande` contenant :

- `id` (identifiant unique),
- `client` (nom du client),
- `priorite` (entier entre 1 et 5, 5 = urgent),
- `duree_estimee` (en minutes).

Une file de demandes est modélisée à l'aide de la classe `CentreAppels`.

```
class Demande:
    def __init__(self, id_demande, client, priorite, duree):
        self.id = id_demande
        self.client = client
        self.priorite = priorite
        self.duree = duree

class CentreAppels:
    def __init__(self):
        self.attente = []

    def ajouter_demande(self, demande):
        self.attente.append(demande)

    def traiter(self):
        if len(self.attente) > 0:
            return self.attente.pop(0)
        return None
```

8. Comment appelle-t-on généralement la méthode `__init__` dans la programmation orientée objet ?

9. Écrire une méthode **`duree_totale()`** de la classe `CentreAppels` qui renvoie la somme des durées estimées des demandes dans `attente`.

10. Écrire les instructions qui créent une instance de `CentreAppels`, ajoute une demande urgente, puis la traite.

11. Compléter la méthode `recherche_prioritaire(p)` qui renvoie **la liste des identifiants des demandes** avec une priorité supérieure ou égale à p.

```
def recherche_prioritaire(self, p):
    ...
    for d in self.attente:
        if ...:
            ...
    ...
```

On souhaite désormais gérer le fonctionnement complet du centre d'appels.
Pour ce faire, on veux traiter toutes les demandes dans l'ordre de leurs priorités.

On laisse le choix de l'implémentation libre.

12. Écrire une fonction lançant le traitement complet, toute trace de recherche sera valorisée.