BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : **3 heures 30**

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

Le sujet comporte **15** pages numérotées de **1/15** à **15/15**
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat traite au choix 3 exercices parmi les 5 exercices proposés

Chaque exercice est noté sur 4 points.

Description des exercices

Cette page décrit sommairement les connaissances abordées dans chacun des exercices de ce sujet. Le candidat traite au choix 3 exercices parmi les 5 exercices proposés.

EXERCICE 1

Cet exercice porte sur un schéma relationnel de bases de données et des requêtes SQL.

EXERCICE 2

Cet exercice porte sur l'architecture des ordinateurs et les protocoles réseau.

EXERCICE 3

Cet exercice est un exercice d'algorithmique et de programmation en langage Python utilisant un fichier csv.

EXERCICE 4

Cet exercice est un exercice d'algorithmique et de programmation en langage Python utilisant les structures de données du type arbre binaire.

EXERCICE 5

Cet exercice est un exercice d'algorithmique et de programmation en langage Python. Il aborde la programmation orientée objet.

EXERCICE 1

Cet exercice porte sur un schéma relationnel de bases de données et des requêtes SQL.

L'Internet Movie Database, noté **IMDb**, est une base de données en ligne sur le cinéma. Cette base de données permet d'accéder à des informations concernant les films, les acteurs, les réalisateurs, les scénaristes, etc. L'accès à ces informations est gratuit.

Une version simplifiée du schéma relationnel de cette base de données est présentée ci-dessous :

Film (Idfilm, titre, annee, #Idrealisateur, noteIMDb)
Acteur (Idacteur, acteur_nom, acteur_prenom)
Joue_Role (Idrole, #Idacteur, #Idfilm, personnage)
Realisateur (Idrealisateur, realisateur_nom, realisateur_prenom)

Dans l'exercice, toutes les requêtes seront écrites en langage SQL.

Les clés primaires sont soulignées et les clés étrangères sont précédées d'un #.

Partie 1 : Étude du schéma relationnel

1) **Indiquer** ce qui différencie les concepts de clé primaire et de clé étrangère.

2) **Indiquer** l'utilité des clés étrangères dans la relation **Joue_Role** de ce schéma relationnel.

Partie 2 : Accès aux données de la base **IMDb**

1) **Écrire** une requête permettant d'obtenir la liste des titres des films et de leur année de sortie.

2) En 2021, Fathia YOUSSOUF a obtenu le César du meilleur espoir féminin pour le film « Mignonnes ».
Écrire une requête permettant de créer l'enregistrement dans la table Acteur de cette actrice selon le modèle suivant :

Idacteur	acteur_nom	acteur_prenom
1250	YOUSSOUF	Fathia

3) On cherche à supprimer un acteur de la base. Une fois la requête correspondante lancée, le message d'erreur suivant apparaît.
Indiquer ce qui a pu provoquer cette erreur.

ERREUR

Requête SQL :

```
DELETE FROM `Acteur` WHERE `acteur_nom`='DUPONT' AND
`acteur_prenom`='David';
```

MySQL a répondu :

1. #1451 - Cannot delete or update a parent row: a foreign key constraint fails (`Joue_Role`, CONSTRAINT `Joue_Role_Acteur_fk` FOREIGN KEY (`Idacteur`) REFERENCES `Acteur` (`Idacteur`))

4) **Écrire** une requête permettant d'obtenir le nom et le prénom du réalisateur du film ayant pour titre « Léon ».

5) **Écrire** une requête permettant d'obtenir le nom des acteurs du film ayant pour titre « Intouchables ».

EXERCICE 2

Cet exercice porte sur l'architecture des ordinateurs et les protocoles réseau.

Partie 1 : Choix d'un serveur

Une microentreprise souhaite s'équiper d'un ordinateur pour héberger son site web. Elle hésite entre deux configurations : un ordinateur monocarte (SoC – System on Chip) Raspberry Pi (figure 1) ou un mini-PC (figure 2).

Figure 1

Raspberry Pi 4B
Processeur ARM 64-bit quad core 1,5 GHz
8 Gio RAM
WiFi et Ethernet
1 prise jack, 4 ports USB, 2 ports micro-HDMI
40 broches GPIO
Carte micro-SD 64 Go (amovible)
Alimentation via USB-C (5 V, 3 A, 15 W)

Figure 2

Mini-PC
Processeur Intel Celeron 64 bit quad core 1,5 GHz
8 Gio RAM
WiFi et Ethernet
2 prises jack, 4 ports USB, 1 port HDMI
SSD 250 Go
Alimentation séparée (19 V, 3,42 A, 65 W)

1) **Indiquer** la quantité de mémoire vive du Raspberry Pi.

2) **Indiquer** la fonction principale du composant SSD (Solid State Drive).

3) **Indiquer** deux avantages d'un ordinateur monocarte (SoC) par rapport à un mini-ordinateur.

Partie 2 : Paramétrage du serveur web

En plus de ce nouvel ordinateur (« serveur web »), un commutateur (« switch ») et un deuxième routeur ont été installés et raccordés pour constituer le « Réseau services ».
Une modélisation de ce nouveau réseau de cette microentreprise est schématisée sur la figure 3.

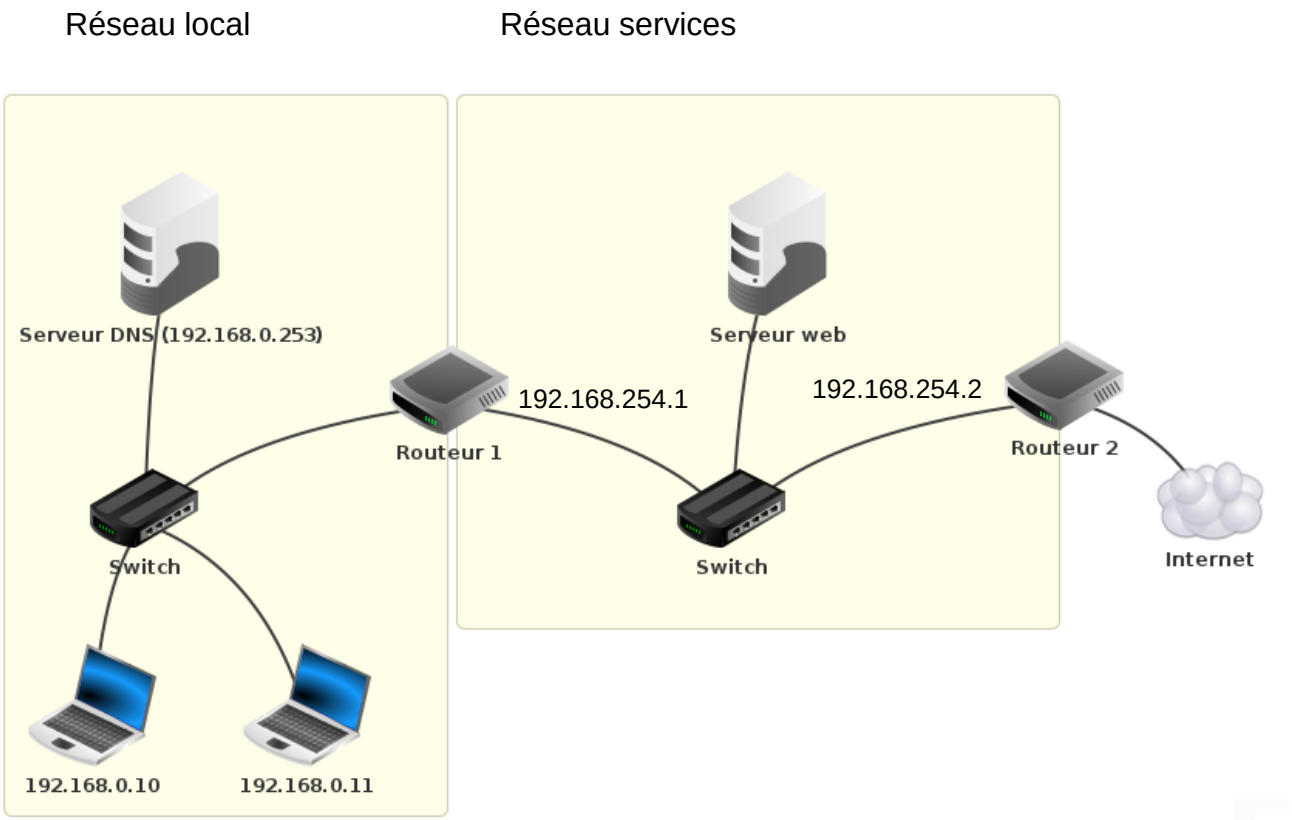


Figure 3

Une adresse IPv4, codée sur 4 octets, doit être associée à un masque de réseau pour être interprétable. Le masque de réseau permet de distinguer la partie de l'adresse qui identifie un réseau de celle qui identifie une machine. Il est codé sur 4 octets soit 32 bits. Dans cet exercice, on utilisera le masque : 255.255.255.0.
Les adresses IP connues sont indiquées. Les routeurs ont deux interfaces, chacune avec son adresse IP.

1) **Indiquer** l'adresse IP du « Réseau services ».

2) **Indiquer** le nombre d'adresses IP encore disponibles qui peuvent être attribuées au serveur web.

3) **Indiquer** l'adresse IP de la passerelle donnant au serveur web l'accès à Internet.

Partie 3 : Analyse d'une trame réseau

La figure 4 représente un extrait d'une capture de trame réseau effectuée au niveau du serveur DNS (Domain Name System), alors que le poste de travail 192.168.0.11 accédait au site web https://www.monsite.sme/ hébergé par le serveur web.

1) En analysant les échanges de trame présentés en figure 4, **indiquer** l'adresse physique / MAC (Media Access Control) du serveur DNS.

No	Source	Destination	Proto.	Couche	Commentaire
1	192.168.0.11	192.168.0.253	ARP	Internet	Adresse MAC de 192.168.0.253 ?
2	192.168.0.253	192.168.0.11	ARP	Internet	192.168.0.253 → 8A:FD:54:49:D0:CC
3	192.168.0.11	192.168.0.253		Application	
4	192.168.0.253	192.168.0.11		Application	

Figure 4

2) En utilisant les détails de la trame N°4 (figure 5) ci-dessous, **indiquer** le protocole utilisé pour cette trame au niveau de la couche transport.

```
No.: 4 / Date: 13:42:11.092
- Réseau
    Source:        8A:FD:54:49:D0:CC
    Destination:   CD:12:79:70:62:6D
    Commentaire:   0x800
- Internet
    Source:        192.168.0.253
    Destination:   192.168.0.11
    Protocole:     IP
    Commentaire:   Protocole :17, TTL: 64
- Transport
    Source:        53
    Destination:   40389
    Protocole:     UDP
- Application
    Commentaire:
        ID=32145 QR=1 RCODE=0 QDCOUNT=0 ANCOUNT=1 NSCOUNT=0 ARCOUNT=0
        www.monsite.sme. A 3600 192.168.254.201
```

Figure 5

3) À partir du détail de la trame N°4 (figure 5), **indiquer** l'adresse IP du serveur web.

Partie 4 : Protocole de routage

Les routeurs 1 et 2, notés respectivement R1 et R2, s'insèrent dans un réseau plus étendu dans lequel les routeurs sont reliés comme indiqué à la figure 6.

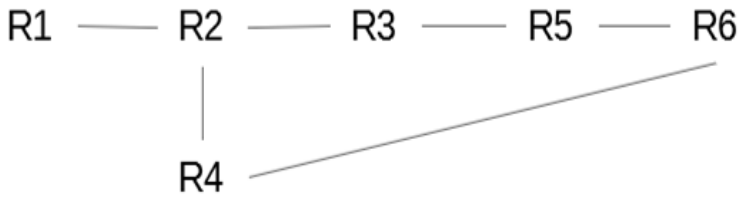


Figure 6

Le protocole RIP (Routing Information Protocol) permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur la distance, en nombre de sauts, qui le sépare d'un autre routeur. La table de routage du routeur R2 est fournie ci-dessous.

Table de routage du routeur R2		
Destination	Routeur suivant	Distance
R1	R1	1
R3	R3	1
R4	R4	1
R5	R3	2
R6	R4	2

1) **Donner** la table de routage du routeur R4.

2) Le routeur R1 doit transmettre un message au routeur R6, en effectuant un nombre minimal de sauts. **Indiquer** le trajet parcouru.

EXERCICE 3

Cet exercice est un exercice d'algorithmique et de programmation en langage Python utilisant un fichier csv.

Partie 1

Dans un fichier villes56.csv, on a regroupé les neuf villes du Morbihan de plus de 10 000 habitants en 2012.

dep	nom	nb_hab_1999	nb_hab_2012	densite	superficie	alt_min	alt_max
56	Lorient	59224	58100	3272	17.48	0	46
56	Vannes	51759	53000	1625	32.3	0	56
56	Lanester	21901	22500	1209	18.37	0	51
56	Ploemeur	18301	18200	448	39.72	-1	55
56	Hennebont	13410	14600	818	18.57	0	82
56	Pontivy	13501	13700	553	24.85	48	192
56	Auray	10899	12500	1783	6.91	0	43
56	Guidel	9155	10100	198	52.29	0	70
56	Saint-Avé	8298	10100	400	26.09	12	136

1) En appliquant le programme suivant :

Programme 1

```
1   import csv
2
3   def readCSV( filename):
4      '''
5      filename est une chaine de caractères correspondant au nom d'un fichier du type csv
6      La fonction renvoie une variable de type list
7      '''
8       with open(filename, mode = 'r', encoding='utf-8', newline = '') as csvFile:
9           fichier = csv.DictReader(csvFile)

10          table1 = [dict(ligne) for ligne in fichier]
11               # les lignes 9 et 10 permettent d'obtenir une liste de dictionnaires dont
    les clés correspondent à la première ligne du fichier csvFile
12      csvFile.close()
13      return table1
14
15  listeVilles = readCSV("villes56.csv")
16  print(listeVilles)
```

a) À partir des quatre propositions ci-dessous, **indiquer** ce que signifie « `utf-8` » à la ligne 8.
Réponse A : type de codage de caractères informatiques.
Réponse B : nom du dossier contenant le fichier.
Réponse C : nom du fichier.
Réponse D : taille du fichier.

b) **Indiquer** ce que signifie le « `mode ='r'` » à la ligne 8.

2) Après exécution du programme Programme 1, voici le début de l'affichage obtenu :

```
[{'dep': '56',
 'nom': 'Lorient',
 'nb_hab_1999': '59224',
 'nb_hab_2012': '58100',
 'densite': '3272',
 'superficie': '17.48',
 'alt_min': '0',
 'alt_max': '46'},
{'dep': '56',……
```

a) **Indiquer** le type de la variable `listeVilles`.

b) **Indiquer** ce que renvoie l'instruction `listeVilles`[1] en précisant son type.

c) **Écrire** une instruction permettant d'obtenir le nombre d'habitants à Pontivy en 2012.

Partie 2

On admet qu'après un traitement, les données figurant dans les colonnes 2 à 7 du tableau `listeVilles` sont des entiers ou des flottants. On considère que `listeVilles` résulte de l'exécution du programme Programme 1.

1) **Indiquer** ce que renvoie l'instruction suivante appliquée à `listeVilles` :
```
>>>[(d['nom'],d['alt_max']) for d in listeVilles if d['alt_max']>100]
```

2) **Écrire** en langage Python une fonction `variations` qui prend en paramètre `listeV`, du même type que la variable `listeVilles`, et renvoie la différence totale entre le nombre d'habitants en 2012 et le nombre d'habitants en 1999 cumulée pour l'ensemble des villes.

3) **Écrire** en langage Python une fonction `surmax` qui prend en paramètre `listeV`, du même type que la variable `listeVilles`, et renvoie la ville ayant la superficie la plus grande. On admet que deux villes n'ont pas la même superficie.

EXERCICE 4

Cet exercice est un exercice d'algorithmique et de programmation en langage Python utilisant les structures de données du type arbre binaire.

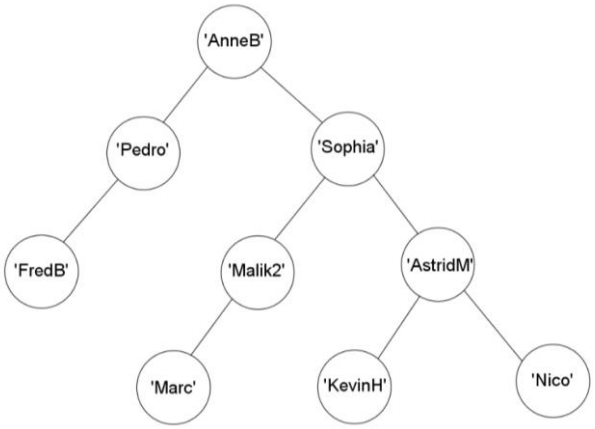
Un club de passionnés d'informatique fonctionne de la façon suivante : pour être membre du club, à l'exception du fondateur, il faut être parrainé. De plus, chaque membre peut parrainer au maximum deux personnes.

Dans ce club, on distingue trois profils de membres :

- membre or : le membre a parrainé deux personnes.
- membre argent : le membre a parrainé une seule personne.
- membre bronze : le membre n'a parrainé personne.

On peut modéliser ce fonctionnement de parrainage à l'aide d'un arbre binaire dont les étiquettes sont les pseudonymes des membres du club.

On donne ci-dessous l'arbre P représentant les membres du club issus des parrainages de AnneB, fondatrice du club. Par exemple, Sophia a parrainé Malik2 et AstridM.



On munit la structure de données arbre binaire des opérations suivantes :

`estvide(Arbre)` renvoie `True` si `Arbre` est vide, `False` sinon

`racine(Arbre)` renvoie l'étiquette du nœud racine de `Arbre`

`gauche(Arbre)` renvoie le sous-arbre gauche de `Arbre`

`droit(Arbre)` renvoie le sous-arbre droit de `Arbre`

1) On définit la hauteur d'un arbre non vide comme la longueur (en nombre d'arêtes) du plus long chemin allant d'une feuille à la racine. Un arbre vide a une hauteur égale à - 1. Un arbre qui n'a qu'un seul nœud a une hauteur de 0.

   a) **Indiquer** la hauteur de l'arbre P.

   b) **Recopier** et **compléter** le programme implémentant la fonction récursive `hauteur` qui prend en argument un arbre binaire `A` et qui renvoie la hauteur de cet arbre. On pourra utiliser la fonction `max` renvoyant la valeur maximale entre deux valeurs.

```
1. def hauteur(A) :
2.      if …… :
3.             return -1
4.      else :
5.             g=hauteur(gauche(A))
6.             d= ……
7.             return 1+ …….
```

   c) **Indiquer** le type de la valeur renvoyée par la fonction `hauteur`.

2) La fonction `membres` ci-dessous prend en argument un arbre binaire `A` et une liste `L`.

```
1. def membres(A,L) :
2.      if not(estvide(A)) :
3.             L.append(racine(A))
4.             membres(gauche(A),L)
5.             membres(droit(A),L)
```

   a) **Écrire** le contenu de `L_membres` après l'exécution des instructions ci-dessous où `P` désigne l'arbre P :

```
L_membres=[ ]
membres(P,L_membres)
```

   b) **Indiquer** le nom du type de parcours d'arbre binaire réalisé par la fonction `membres`.

3) Dans cette question, on s'intéresse aux profils des membres (or, argent ou bronze).

   a) **Indiquer** les étiquettes des feuilles de l'arbre P.

   b) À partir des propositions suivantes, **indiquer** le profil des membres dont les pseudonymes sont les étiquettes (définies en début de sujet) des feuilles.
   Réponse A : membre or.
   Réponse B : membre argent.
   Réponse C : membre bronze.
   Réponse D : on ne peut pas savoir.

c) **Écrire** la fonction `profil` prenant en argument un arbre binaire non vide A et qui renvoie le profil du membre dont le pseudonyme est l'étiquette de la racine de A, sous la forme d'une chaine de caractères : '`membre_or`', '`membre_argent`' ou '`membre_bronze`'. Par exemple, l'appel à `profil(P)` doit renvoyer '`membre_or`' qui correspond au profil du membre 'AnneB', racine de P.

4) Afin d'obtenir un tableau dont chaque élément est un tuple contenant le pseudonyme d'un membre et son profil, on propose la fonction `membres_profiles` définie ci-dessous :

```
1. def membres_profiles(A,L) :
2.       if not(estvide(A)) :
3.              L.append((racine(A),profil(A)))
4.              membres_profiles(gauche(A),L)
5.              membres_profiles(droit(A),L)
```

On exécute les instructions ci-dessous :

```
L2=[ ]
membres_profiles(arbre2,L2)
```

À l'issue de ces instructions on a obtenu le tableau suivant :

```
L2=[('LeaC','membre_or'),('Ali','membre_bronze'),('Tom45','membre_argent'),('Vero','membre_bronze')]
```

Dessiner un arbre possible pouvant correspondre à `arbre2`.

5) Chaque année, les membres reversent une cotisation en fonction de leur profil.

- membre or : cotisation de 20 €
- membre argent : cotisation de 30 €
- membre bronze : cotisation de 40 €

Écrire une fonction `cotisation` en langage Python ou en pseudo-code qui prend en argument un arbre binaire modélisant les parrainages comme explicité au début de l'exercice et qui renvoie le montant total des cotisations reçues par le club. On pourra utiliser la fonction `membres_profiles` de la question précédente.

EXERCICE 5

Cet exercice est un exercice d'algorithmique et de programmation en langage Python. Il aborde la programmation orientée objet.

Des élus d'un canton français décident de mettre en place une monnaie locale complémentaire dans leur circonscription, appelée le « sou ». L'objectif est d'encourager la population à acheter auprès de vendeurs et producteurs locaux.

La plus petite valeur du sou est le billet de 1 sou, il ne peut donc pas être séparé en dessous de ce montant. Le sou a un taux de change à parité avec l'euro (1 sou = 1 euro), de façon à ce que le prix d'un article puisse être intégralement payé en sous. Si le prix d'un article n'est pas entier, la différence peut être payée en euros. Par exemple, un article coûtant 3,50 € peut être payé avec 3 sous et 50 centimes d'euros.

Le canton a créé une association chargée de gérer les comptes de ses futurs adhérents au moyen d'une application en langage Python. On a commencé à créer une classe `Compte` dont les propriétés sont : le numéro de compte, le nom de l'adhérent, son adresse et le solde du compte. Lors de la création d'un compte, il faudra renseigner toutes les propriétés sauf le solde qui sera mis à 0 sou. Les méthodes de cette classe sont les suivantes :

Méthodes de la classe `Compte`	Description
`__init__(self, numero, adherent, adresse)`	Crée un nouveau compte et met le solde à 0.
`crediter(self, montant)`	Ajoute montant au solde.
`debiter(self, montant)`	Enlève montant au solde.
`est_positif(self)`	Renvoie Vrai si le solde du compte est positif ou nul.

Partie 1 : Création de la classe `compte`

On a commencé à écrire la classe `Compte`.

```
class Compte:
    def __init__(self, numero, adherent, adresse):
        self.numero = numero
        self.adherent = adherent
        self.adresse = adresse
        self.solde = 0
```

1) **Écrire** sur la copie la méthode `crediter`.
2) **Écrire** sur la copie la méthode `debiter`.
3) **Écrire** sur la copie la méthode `est_positif`.

Partie 2 : Utilisation de la classe `compte`

Monsieur Martin souhaite rejoindre la communauté des utilisateurs du sou et déposer 200 € sur son compte en sou.

1) **Écrire** la ligne de code en langage Python permettant de créer le compte `cpt_0123456A` dont le numéro est "`0123456A`", le titulaire est "`MARTIN Dominique`" qui habite à l'adresse "`12 rue des sports`".

2) **Écrire** la ligne de code en langage Python permettant de déposer 200 € sur le compte de monsieur Martin.

3) Monsieur Martin souhaite transférer 50 sous de son compte « `cpt_0123456A` » vers un autre compte « `cpt_2468794E` ». On a besoin pour cela d'ajouter une méthode à la classe `Compte`.

   Écrire la méthode `transferer`(self, `autre_compte`, `montant`) qui transfère le montant passé en paramètre vers un autre compte. On suppose que le compte courant est suffisamment approvisionné. On utilisera les autres méthodes de la classe `Compte` sans en modifier directement les attributs.

Partie 3 : Gestion des comptes

Pour en faciliter la gestion, on crée une liste Python `liste_comptes` contenant tous les comptes des adhérents. Cette liste sera donc composée d'objets de type `compte`.

Le gestionnaire de l'association souhaite relancer les adhérents dont le compte est débiteur, c'est-à-dire dont le solde est négatif. Il faut, pour cela, ajouter une autre fonction au programme en langage Python.

1) **Écrire** la fonction `rechercher_debiteurs` qui prend en argument `liste_comptes` et renvoie une liste de dictionnaires dont la première clé est le nom de l'adhérent et la seconde clé le solde de son compte en négatif.

   Exemple de résultat :
   `liste_debiteurs = [{'Nom': 'DUPONT Thomas', 'solde': -60}, {'Nom': 'CARNEIRO Sarah', 'solde': -75}]`.

2) **Écrire** la fonction `urgent_debiteur` qui prend en argument la liste de dictionnaires et renvoie le nom de l'adhérent le plus endetté. On suppose qu'aucun adhérent n'a la même dette.