BACCALAURÉAT GÉNÉRAL BLANC 2024

NSI

Epreuve de spécialité

DURÉE DE L'ÉPREUVE : **3 h 30**

L'usage d'une calculatrice N´EST PAS autorisé

Exercice 1: **RESEAUX, PROCESSUS ET SYSTEMES D´EXPLOITATION**
Exercice 2: **ARBRES, FILES ET POO**
Exercice 3: **SQL, DICTIONNAIRES ET POO**

L´ÉPREUVE EST NOTEE SUR 30 POINTS

EXERCICE 1: RESEAUX, PROCESSUS ET SYSTEMES D´EXPLOITATION (11,5 POINTS)

Cet exercice est constitué de deux parties indépendantes.

PARTIE 1 :

Cette partie traite du thème architecture matérielle, et plus particulièrement des processus et leur ordonnancement.

1. Avec la commande `ps –aef` on obtient l'affichage suivant :

```
PID  PPID  C STIME TTY          TIME CMD
8600     2  0 17:38 ?        00:00:00 [kworker/u2:0-fl]
8859     2  0 17:40 ?        00:00:00 [kworker/0:1-eve]
8866     2  0 17:40 ?        00:00:00 [kworker/0:10-ev]
8867     2  0 17:40 ?        00:00:00 [kworker/0:11-ev]
8887  6217  0 17:40 pts/0    00:00:00 bash
9562     2  0 17:45 ?        00:00:00 [kworker/u2:1-ev]
9594     2  0 17:45 ?        00:00:00 [kworker/0:0-eve]
9617  8887 21 17:46 pts/0    00:00:06 /usr/lib/firefox/firefox
9657  9617 17 17:46 pts/0    00:00:04 /usr/lib/firefox/firefox -contentproc -childID
9697  9617  4 17:46 pts/0    00:00:01 /usr/lib/firefox/firefox -contentproc -childID
9750  9617  3 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox -contentproc -childID
9794  9617 11 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox -contentproc -childID
9795  9794  0 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox
9802  7441  0 17:46 pts/2    00:00:00 ps -aef
```

On rappelle que :

`PID` = Identifiant d'un processus (Process Identification)

`PPID` = Identifiant du processus parent d'un processus (Parent Process Identification)

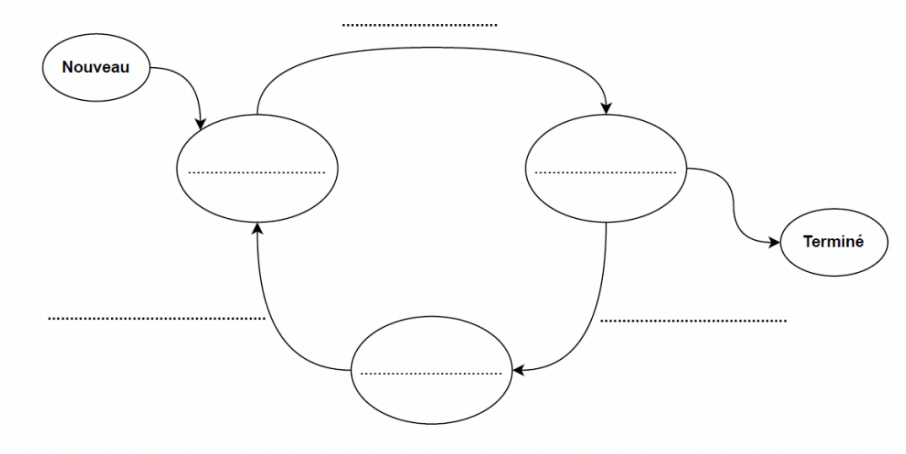
a. Donner sous forme d'un arbre de `PID` la hiérarchie des processus liés à `firefox`.

b. Indiquer la commande qui a lancé le premier processus de `firefox`.

c. La commande `kill` permet de supprimer un processus à l'aide de son `PID` (par exemple `kill 8600`). Indiquer la commande qui permettra de supprimer tous les processus liés à `firefox` et uniquement cela.

2.

a. Recopier et compléter le schéma ci-dessous avec les termes suivants concernant l'ordonnancement des processus : *Élu, En attente, Prêt, Blocage, Déblocage, Mise en exécution*.



On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivée et une durée, donnés en nombre de cycles du processeur.

Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

Les processus sont placés dans une file d'attente en fonction de leur instant d'arrivée.
On se propose d'ordonnancer ces quatre processus avec la méthode suivante :

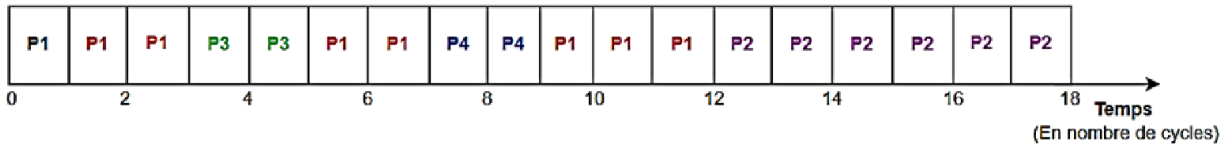
➢ Parmi les processus présents en liste d'attente, l'ordonnanceur choisit celui dont la **durée restante** est la plus courte ;

➢ Le processeur exécute un cycle de ce processus puis l'ordonnanceur désigne de nouveau le processus dont la durée restante est la plus courte ;

➢ En cas d'égalité de temps restant entre plusieurs processus, celui choisi sera celui dont l'instant d'arrivée est le plus ancien ;

➢ Tout ceci jusqu'à épuisement des processus en liste d'attente.

On donne en exemple ci-dessous, l'ordonnancement des quatre processus de l'exemple précédent suivant l'algorithme ci-dessus.

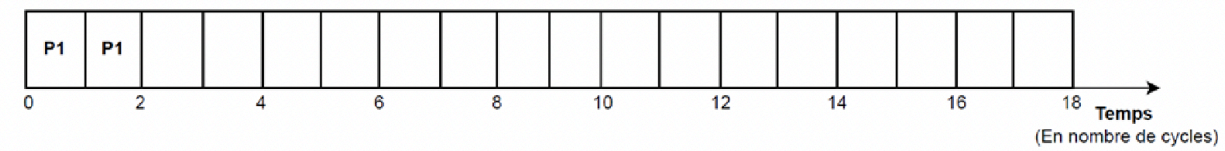


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

b. Calculer la moyenne des temps d'exécution des quatre processus.

On se propose de modifier l'ordonnancement des processus. L'algorithme reste identique à celui présenté précédemment mais au lieu d'exécuter un seul cycle, le processeur exécutera à chaque fois deux cycles du processus choisi. En cas d'égalité de temps restant, l'ordonnanceur départagera toujours en fonction de l'instant d'arrivée.

c. Recopier et compléter le schéma ci-dessous donnant le nouvel ordonnancement des quatre processus.



d. Calculer la nouvelle moyenne des temps d'exécution des quatre processus et indiquer si cet ordonnancement est plus performant que le précédent.

3. On se propose de programmer l'algorithme du premier ordonnanceur. Chaque processus sera représenté par une liste comportant autant d'éléments que de durées (en nombre de cycles). Pour simuler la date de création de chaque processus, on ajoutera en fin de liste de chaque processus autant de chaînes de caractères vides que la valeur de leur date de création.

```
1  p1 = ['1.8','1.7','1.6','1.5','1.4','1.3','1.2','1.1']
2  p2 = ['2.6','2.5','2.4','2.3','2.2','2.1','','']
3  p3 = ['3.2','3.1','','','']
4  p4 = ['4.2','4.1','','','','','','','']
5  liste_proc = [p1, p2, p3, p4]
```

La fonction `choix_processus` est chargée de sélectionner le processus dont le temps restant d'exécution est le plus court parmi les processus en liste d'attente.

a. Recopier sans les commentaires et compléter la fonction `choix_processus` ci-dessous. Le code peut contenir plusieurs lignes.

```
1  def choix_processus(liste_attente):
2      """Renvoie l'indice du processus le plus court parmi
3      ceux présents en liste d'attente liste_attente"""
4      if liste_attente != []:
5          mini = len(liste_attente[0])
6          indice = 0
7          ...
8          return indice
```

Une fonction `scrutation` (non étudiée) est chargée de parcourir la liste `liste_proc` de tous les processus et de renvoyer la liste d'attente des processus en fonction de leur arrivée. À chaque exécution de `scrutation`, les processus présents (sans chaînes de caractères vides en fin de liste) sont ajoutés à la liste d'attente. La fonction supprime pour les autres un élément de chaîne de caractères vides.

b. Recopier et compléter les différentes instructions de la fonction `ordonnancement` pour réaliser le fonctionnement désiré.

```
1  def ordonancement(liste_proc):
2      """Exécute l'algorithme d'ordonnancement
3      liste_proc -- liste des processus
4      Renvoie la liste d'exécution des processus"""
5      execution = []
6      attente = scrutation(liste_proc, [])
7      while attente != []:
8          indice = choix_processus(attente)
9          ... # A FAIRE (plusieurs lignes de code) ...
10         attente = scrutation(liste_proc, attente)
11     return execution
```

PARTIE 2 :

Cette partie porte sur l'architecture matérielle, les réseaux et les systèmes d'exploitation.

Nous allons étudier les communications entre Bob et Alice. Ils communiquent au travers du réseau ci-dessous dont le protocole de routage est le protocole OSPF qui minimise le coût des communications :

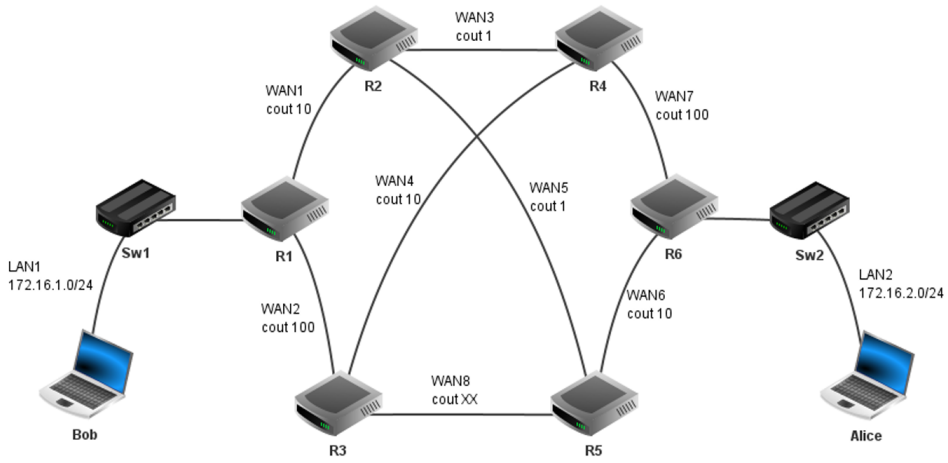


Figure 1 : Plan du réseau de communication entre Alice et Bob.
LAN : réseau local ; WAN : réseau étendu ; R : routeur ; Sw : Switch

Une adresse IPv4 est composée de quatre octets soit 32 bits. Une adresse de sous-réseau avec la notation /n signifie que les n premiers bits de l'adresse correspondent à la partie «réseau» et les suivants à la partie «machine».

L'adresse dont tous les bits de la partie « machine » sont à 0 est appelée adresse du réseau.
L'adresse dont tous les bits de la partie « machine » sont à 1 est appelée adresse de diffusion.
Ces adresses sont réservées et ne peuvent pas être attribuées à des machines.
Le choix des routes empruntées par les paquets IP est uniquement basé sur le protocole OSPF.
On prendra comme débit maximal de référence `10 000 Mbit/s.`

Le coût est alors calculé de la façon suivante :

$$\textbf{coût} = \frac{\textbf{débit maximal de référence}}{\textbf{débit du réseau concerné}}$$

4.

a. La configuration IP partielle ci-dessous a été affichée sur l'un des ordinateurs :

```
IP hôte : 172.16.2.3
IP passerelle : 172.16.2.253
```

Indiquer en justifiant si cette configuration appartient à l'ordinateur de Bob ou d'Alice.

b. Combien peut-on brancher en tout de machines sur ce réseau ? Expliquer.

5. Le réseau WAN8 a un débit de `1 000 Mbit/s`. Calculer le coût correspondant.

6. On donne les tables de routage des routeurs R1 à R5, dans lesquelles `Pass.` désigne la passerelle (qui correspond au routeur suivant) :

Routeur R1		
Destination	Pass.	Coût
LAN1	-	-
LAN2	R2	21
WAN1	-	-
WAN2	-	-
WAN3	R2	10
WAN4	R2	11
WAN5	R2	10
WAN6	R2	11
WAN7	R2	11
WAN8	R2	11

Routeur R2		
Destination	Pass.	Coût
LAN1	R1	10
LAN2	R5	11
WAN1	-	-
WAN2	R1	10
WAN3	-	-
WAN4	R4	1
WAN5	-	-
WAN6	R5	1
WAN7	R4	1
WAN8	R5	1

Routeur R3		
Destination	Pass.	Coût
LAN1	R4	21
LAN2	R5	20
WAN1	R4	11
WAN2	-	-
WAN3	R4	10
WAN4	-	-
WAN5	R5	10
WAN6	R5	10
WAN7	R4	10
WAN8	-	-

Routeur R4		
Destination	Pass.	Coût
LAN1	R2	11
LAN2	R2	12
WAN1	R2	1
WAN2	R3	10
WAN3	-	-
WAN4	-	-
WAN5	R2	1
WAN6	R2	2
WAN7	-	-
WAN8	R2	2

Routeur R5		
Destination	Pass.	Coût
LAN1	R2	11
LAN2	R6	10
WAN1	R2	1
WAN2	R3	10
WAN3	R2	1
WAN4	R2	2
WAN5	-	-
WAN6	-	-
WAN7	R2	2
WAN8	-	-

Figure 2 : Tables de routage des routeurs R1 à R5

Écrire sur votre copie la table de routage du routeur R6.

7. Bob envoie un message à Alice. Énumérer dans l'ordre tous les routeurs par lesquels transitera ce message.

8. Un routeur tombe en panne, le nouveau `coût` pour la route entre Bob et Alice est de 111. Déterminer le nom du routeur en panne.

EXERCICE 2: ARBRES, FILES ET POO (7 POINTS)

Cet exercice porte sur les arbres binaires, les files et la programmation orientée objet.
Cet exercice comporte deux parties indépendantes.

PARTIE 1

Une entreprise stocke les identifiants de ses clients dans un arbre binaire de recherche. On rappelle qu'un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit.

La taille d'un arbre est le nombre de nœuds qu'il contient. Sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles. On convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1 et celle de l'arbre vide vaut 0.

Dans cet arbre binaire de recherche, chaque nœud contient une valeur, ici une chaine de caractères, qui est, avec l'ordre lexicographique (celui du dictionnaire) :

– strictement supérieure à toutes les valeurs des nœuds du sous-arbre gauche ;
– strictement inférieure à toutes les valeurs des nœuds du sous-arbre droit.

Ainsi les valeurs de cet arbre sont toutes distinctes.

On considère l'arbre binaire de recherche suivant :

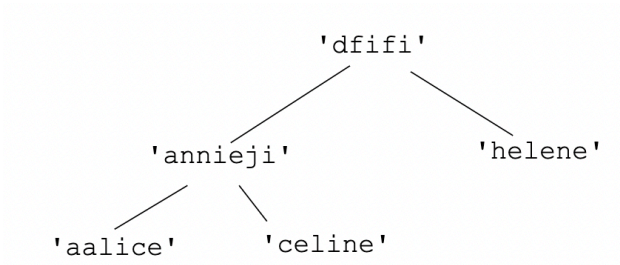


Figure 1. Arbre binaire de recherche.

1. Donner la taille et la hauteur de l'arbre binaire de recherche de la figure 1.
2. Recopier cet arbre après l'ajout des identifiants suivants : `'davidbg '` et `'papicoeur'` dans cet ordre.
3. On décide de parcourir cet arbre pour obtenir la liste des identifiants dans l'ordre lexicographique. Recopier la lettre correspondant au parcours à utiliser parmi les propositions suivantes :

A - Parcours en largeur d'abord
B - Parcours en profondeur dans l'ordre préfixe
C - Parcours en profondeur dans l'ordre infixe
D - Parcours en profondeur dans l'ordre suffixe (ou postfixe)

4. Pour traiter informatiquement les arbres binaires, nous allons utiliser une classe `ABR`.
Un arbre binaire de recherche, nommé `abr` dispose des méthodes suivantes :

– `abr.est_vide()`: renvoie True si `abr` est vide et False sinon.
– `abr.racine()`: renvoie l'élément situé à la racine de `abr` si `abr` n'est pas vide et `None` sinon.
– `abr.sg()`: renvoie le sous-arbre gauche de `abr` s'il existe et `None` sinon.
– `abr.sd()`: renvoie le sous-arbre droit de `abr` s'il existe et `None` sinon.

On a commencé à écrire une méthode récursive `present` de la classe `ABR`, où le paramètre `identifiant` est une chaine de caractères et qui retourne `True` si `identifiant` est dans l'arbre et `False` sinon.

```
1    def present(self, identifiant):
2        if self.est_vide():
3            return False
4        elif self.racine() == identifiant:
5            return ...
6        elif self.racine() < identifiant:
7            return self.sd(). ...
8        else:
9            return ...
```

Recopier et compléter les lignes 5, 7 et 9 de cette méthode.

PARTIE 2

On considère une structure de données file que l'on représentera par des éléments en ligne, l'élément à droite étant la tête de la file et l'élément à gauche étant la queue de la file.
On appellera `f1` la file suivante :

	'bac'	'nsi'	'2023'	'file'	

On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python;

- `creer_file()` : renvoie une file vide ;
- `est_vide(f)` : renvoie True si la file `f` est vide et `False` sinon ;
- `enfiler(f, e)` : ajoute l'élément `e` à la queue de la file `f` ;
- `defiler(f)` : renvoie l'élément situé à la tête de la file `f` et le retire de la file.

5. Les trois questions suivantes sont indépendantes.

a. Donner le résultat renvoyé après l'appel de la fonction `est_vide(f1)`.

b. Représenter la file `f1` après l'exécution du code `defiler(f1)`.

c. Représenter la file `f2` après l'exécution du code suivant :

```
1  f2 = creer_file()
2  liste = ['castor', 'python', 'poule']
3  for elt in liste:
4      enfiler(f2, elt)
```

6. Recopier et compléter les lignes 4, 6 et 7 de la fonction `longueur` qui prend en paramètre une file `f` et qui renvoie le nombre d'éléments qu'elle contient. Après un appel à la fonction, la file `f` doit retrouver son état d'origine.

```
1   def longueur(f):
2       resultat = 0
3       g = creer_file()
4       while ... :
5           elt = defiler(f)
6           resultat = ...
7           enfiler(... , ...)
8       while not(est_vide(g)):
9           enfiler(f, defiler(g))
10      return resultat
```

7. Un site impose à ses clients des critères sur leur mot de passe. Pour cela il utilize la fonction `est_valide` qui prend en paramètre une chaine de caractères `mot` et qui retourne `True` si `mot` correspond aux critères et `False` sinon.

```
1  def est_valide(mot):
2      if len(mot) < 8:
3          return False
4      for c in mot:
5          if c in ['!', '#', '@', ';', ':']:
6              return True
7      return False
```

Parmi les mots de passe suivants, recopier celui qui sera validé par cette fonction.

A – `'best@ '`
B – `'paptap23 '`
C – `'2!@59fgds '`

8. Le tableau suivant montre, sur deux exemples, l'évolution d'une file `f3` après l'exécution de l'instruction `ajouter_mot(f3, 'super ')` :

	état initial de `f3`	**état de `f3` après l'instruction `ajouter_mot(f3, 'super')`**
Exemple 1	`'bac'` `'nsi'` `'2023'`	`'super'` `'bac'` `'nsi'`
Exemple 2	`'test'` `'info'`	`'super'` `'test'` `'info'`

Écrire le code de cette fonction `ajouter_mot` qui prend en paramètres une file `f` (qui a au plus 3 éléments) et une chaine de caractères valide `mdp`. Cette fonction met à jour la file de stockage `f` des mots de passe en y ajoutant `mdp` et en défilant, si nécessaire, pour avoir au maximum trois éléments dans cette file.
On pourra utiliser la fonction `longueur` de la question 6.

9. Pour intensifier sa sécurité, le site stocke les trois derniers mots de passe dans une file et interdit au client lorsqu'il change son mot de passe d'utiliser l'un des mots de passe stockés dans cette file.

Recopier et compléter les lignes 7 et 8 de la fonction `mot_file` :
- qui prend en paramètres une file `f` et `mdp` de type chaine de caractères ;
- qui renvoie `True` si le mot de passe est un élément de la file `f` et `False` sinon.

Après un appel à cette fonction, la file `f` doit retrouver son état d'origine.

```
1   def mot_file(f, mdp):
2       g = creer_file()
3       present = False
4       while not(est_vide(f)):
5           elt = defiler(f)
6           enfiler(g, elt)
7           if ...:
8               present = ...
9       while not(est_vide(g)):
10          enfiler(f, defiler(g))
11      return present
```

10. Écrire une fonction `modification` qui prend en paramètres une file `f` et une chaine de caractères `nv_mdp`. Si le mot de passe `nv_mdp` répond bien aux **deux** exigences des questions 7 et 9, alors elle modifie la file des mots de passe stockés et renvoie `True`. Dans le cas contraire, elle renvoie `False`.

On pourra utiliser les fonctions `mot_file`, `est_valide` et `ajouter_mot`.

EXERCICE 3: SQL, DICTIONNAIRES ET POO (11,5 POINTS)

Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.
Cet exercice est composé de 3 parties indépendantes.

On veut créer une application permettant de stocker et de traiter des informations sur des livres de science-fiction. On désire stocker les informations suivantes :

- l'identifiant du livre (`id`) ;
- le titre (`titre`) ;
- le nom de l'auteur (`nom_auteur`) ;
- l'année de première publication (`ann_pub`) ;
- une note sur 10 (`note`).

Voici un extrait des informations que l'on cherche à stocker :

Livres de science-fiction				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
1 dico_livres = {
    'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
    'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade Runner',
        'Les Robots', 'Ravage', 'Chroniques martiennes',
        'Dragon déchu', 'Fahrenheit 451'],
    'auteur' : ['Orwell', 'Herbert', 'Asimov',
        'K.Dick', 'K.Dick', 'Asimov', 'Barjavel',
        'Bradbury', 'Hamilton', 'Bradbury'],
    'ann_pub' : [1949, 1965, 1951, 1953,1968,
        1950, 1943, 1950, 2003, 1953],
    'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
  }

2 a = dico_livres['note']
3 b = dico_livres['titre'][2]
```

1. Déterminer les valeurs des variables `a` et `b` après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None.`

```
1 def titre_livre(dico, id_livre):
2     for i in range(len(dico['id'])):
3         if dico['id'][i] == ... :
4             return dico['titre'][...]
5     return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre.`

3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.

4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).

5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Partie B

Dans cette partie, on utilise la programmation orientée objet (POO). On propose deux classes : `Livre` et `Bibliotheque`.

```
1  class Livre:
2      def __init__(self, id_livre, titre, auteur, ann_pub, note):
3          self.id = id_livre
4          self.titre = titre
5          self.auteur = auteur
6          self.ann_pub = ann_pub
7          self.note = note
8      def get_id(self):
9          return self.id
10     def get_titre(self):
11         return self.titre
12     def get_auteur(self):
13         return self.auteur
14     def get_ann_pub(self):
15         return self.ann_pub
16
17 class Bibliotheque:
18     def __init__(self):
19         self.liste_livre = []
20     def ajout_livre(self, livre):
21        self.liste_livre.append(livre)
22     def titre_livre(self, id_livre):
23         for livre in self.liste_livre :
24             if ... == id_livre :
25                 return ...
26         return ...
```

6. Citer un attribut et une méthode de la classe `Livre`.

7. Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra renvoyer la note d'un livre.

8. Écrire le programme permettant d'ajouter le livre Blade Runner à la fin de la "bibliothèque" en utilisant la classe `Livre` et la classe `Bibliotheque` (voir le tableau en début d'exercice).

9. Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque.`
Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table livres. Cette table livres contient toutes les données sur les livres. On obtient donc la table suivante :

livres				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

L'attribut `id` est la clé primaire pour la table `livres`.

10. Expliquer pourquoi l'attribut `auteur` ne peut pas être choisi comme clé primaire.

11. Donner le résultat renvoyé par la requête SQL suivante :

```
SELECT titre
FROM livres
WHERE auteur = 'K.Dick';
```

12. Écrire une requête SQL permettant d'obtenir les titres des livres écrits par Asimov publiés après 1950.

13. Écrire une requête SQL permettant de modifier la note du livre Ubik en la passant de 9/10 à 10/10.

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type INT ;
- `nom` de type TEXT ;
- `prenom` de type TEXT ;
- `annee_naissance` de type INT (année de naissance).

auteurs			
id	nom	prenom	annee_naissance
1	Orwell	George	1903
2	Herbert	Franck	1920
3	Asimov	Isaac	1920
4	K.Dick	Philip	1928
5	Bradbury	Ray	1920
6	Barjavel	René	1911
7	Hamilton	Peter	1960

La table `livres` est aussi modifiée comme suit :

livres				
id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	6	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

14. Expliquer l'intérêt d'utiliser deux tables (`livres` et `auteurs`) au lieu de regrouper toutes les informations dans une seule table.
15. Expliquer le rôle de l'attribut `id_auteur` de la table `livres`.
16. Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés après 1960.
17. Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance < 30;
```

Un élève décide de créer une application d'annuaire pour sa classe. On pourra retrouver, grâce à cette application, différentes informations sur les élèves de la classe : nom, prénom, date de naissance, numéro de téléphone, adresse email, etc.

18. Expliquer en quoi la réalisation de ce projet pourrait être problématique.