#### BACCALAURÉAT GÉNÉRAL

##### ÉPREUVE D’ENSEIGNEMENT DE SPÉCIALITÉ

###### **SESSION 2025**

#### **NUMÉRIQUE ET SCIENCES INFORMATIQUES**

###### **JOUR 2**

##### 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 14 pages numérotées de 1/14 à 14/14.

#####

**Le sujet est composé de trois exercices indépendants.**

**Le candidat traite les trois exercices.**

# EXERCICE 1 (6 points)

*Cet exercice porte sur la programmation orientée objet et l’algorithmique.*

Pour travailler sur des dates, on a créé la classe Date dont le code est écrit ci-dessous :

1 class Date:
2 def \_\_init\_\_(self, jour, mois, annee):
3 self.jour = ...
4 self.mois = ...
5 self.annee = ...
6 self.nb\_jours\_par\_mois = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
7
8
9 def get\_jour(self):
10 return self.jour
11
12 def get\_mois(self):
13 return self.mois
14
15 def get\_annee(self):
16 return ...
17
18 def set\_jour(self, jour):
19 self.jour = jour
20
21 def set\_mois(self, mois):
22 self.mois = ...
23
24 def set\_annee(self, annee):
25 self.annee = annee
26
27 def est\_bissextile(self):
28 ...

## Partie A : Accès et modification des données

Le constructeur de la classe Date prend en paramètres trois entiers représentant le jour, le mois et l’année, puis les affecte respectivement aux attributs jour, mois et annee.

1. Recopier et compléter les lignes 3 à 5 du code précédent.
2. Indiquer à quelle date correspond l’instance de la classe Date suivante :

d = Date(1, 5, 2000)

1. Écrire le code permettant de créer une instance d de la classe Date qui représente la date du 19 juin 2024.
2. La méthode get\_annee renvoie la valeur de l’attribut annee.
Recopier et compléter les lignes 15 et 16 du code précédent.
3. La méthode set\_mois modifie l’attribut mois en lui affectant la valeur passée en argument.

Recopier et compléter les lignes 21 et 22 du code précédent.

L’attribut nb\_jours\_par\_mois contient une liste qui correspond au nombre de jours pour chaque mois. Le mois de février contient généralement 28 jours mais lors des années bissextiles il en contient 29.

1. La classe Date dispose d’une méthode est\_bissextile, qui utilise uniquement l’attribut annee, et qui renvoie True si l’année de l’instance courante est bissextile et False sinon. On veut compléter la méthode \_\_init\_\_ pour ajuster le nombre de jours par mois pour les années bissextiles.

Recopier et compléter les lignes 7 et 8 suivantes :

6 self.nb\_jours\_par\_mois = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
7 if ... :
8 self.nb\_jours\_par\_mois[...] = 29

## Partie B : sur l’année de l’instance courante

Pour déterminer le nombre de jours au cours d’une année, il faut savoir si elle est bissextile.

Une année est bissextile si elle est divisible par 4 mais pas par 100 ou si elle est divisible par 400.

1. Écrire le code de la méthode est\_bissextile.
On rappelle qu’un entier a est divisible par l’entier n si a % n == 0.

On dote la classe Date de la méthode nb\_jours\_passes qui renvoie le nombre de jours passés dans l’année de l’instance courante.

def nb\_jours\_passes(self):
 nb\_jours = self.jour
 mois = self.mois - 2
 while mois >= 0:
 nb\_jours = nb\_jours + self.nb\_jours\_par\_mois[mois]
 mois = mois - 1
 return nb\_jours

1. Indiquer quel sera l’affichage en console après l’exécution des deux instructions suivantes :

>>> d1 = Date(20, 3, 2001)
>>> d1.nb\_jours\_passes()

On dote la classe Date de la méthode nb\_jours\_restants qui renvoie le nombre de jours restants dans l’année de l’instance courante, soit 366 ou 365 moins le nombre de jours déjà passés, selon que l’année est bissextile ou non.

1. Recopier et compléter le script de la méthode nb\_jours\_restants ci-après :

def nb\_jours\_restants(self):
 j = 365
 if ...:
 j = 366
 return j - ...

## Partie C : entre deux dates

On dote la classe Date de la méthode nb\_jours\_depuis qui prend en paramètre une autre instance other de la classe Date et qui renvoie le nombre de jours écoulés entre la date de l’instance other et la date de l’instance courante.

def nb\_jours\_depuis(self, other):
 if other.get\_annee() > self.get\_annee():
 return -1
 if other.get\_annee() == self.get\_annee():
 if other.nb\_jours\_passes() > self.nb\_jours\_passes() :
 return -1
 if other.nb\_jours\_passes() == self.nb\_jours\_passes() :
 return 0
 nb\_jours = self.nb\_jours\_passes() + other.nb\_jours\_restants()
 for annee in range(other.get\_annee()+1, self.get\_annee()):
 d\_suivant = date(1, 1, annee)
 if d\_suivant.est\_bissextile():
 nb\_jours += 366
 else:
 nb\_jours += 365
 return nb\_jours

On crée les instances de la classe Date suivantes :

>>> d1 = Date(15, 6, 2024)
>>> d2 = Date(15, 6, 2024)
>>> d3 = Date(15, 7, 2024)
>>> d4 = Date(15, 6, 2025)
>>> d5 = Date(15, 6, 2022)

1. Indiquer quels seront alors les affichages en console après l’exécution de chacune des instructions suivantes (on précise que l’année 2024 est bissextile) :

>>> d1.nb\_jours\_depuis(d2)
>>> d1.nb\_jours\_depuis(d3)
>>> d1.nb\_jours\_depuis(d4)
>>> d1.nb\_jours\_depuis(d5)

Le timestamp est le nombre de secondes qui se sont écoulées depuis le 1er janvier 1970 à 00h00. Il s’agit de la date de la mise en marche du système d’exploitation UNIX.

Par exemple, le 01/01/2024 à 00:00:00 correspond au timestamp de 1704063600.

1. Recopier et compléter le code de la méthode timestamp qui renvoie le nombre de secondes qui se sont écoulées depuis le 1er janvier 1970.

1 def timestamp(self):
2 d = ...
3 return self.nb\_jours\_depuis(d) \* 24 \* 3600

# EXERCICE 2 (6 points)

*Cet exercice porte sur la programmation Python, la gestion des processus.*

On souhaite élaborer un programme système permettant de gérer l’ordre d’exécution des processus sur le processeur.

1. Donner le nom de ce type de programme.
2. Donner les différents états possibles d’un processus.

Chaque processus dispose d’une valeur de priorité. Un processus est prioritaire sur un autre processus si sa valeur de priorité est plus petite. Ainsi pour rendre un processus moins prioritaire, il faut augmenter sa valeur de priorité, par exemple en la faisant passer de 2 à 3.

Fonctionnement du programme gérant l’ordre d’exécution des processus :

On dispose d'une liste dont les éléments sont des files de processus. La première file contient les processus ayant la valeur de priorité la plus élevée 0, la seconde ceux ayant la valeur de priorité 1, etc.
 À l'arrivée d'un nouveau processus :
 • Attribuer au nouveau processus la valeur de priorité la plus élevée 0 ;
 • Placer le nouveau processus dans la file d'attente correspondant à sa valeur de priorité (c’est-à-dire la première file de la liste).

 À chaque cycle d'horloge :
 • S'il n'y a pas de processus en cours d'exécution et s'il reste des processus en attente :
 \* Sélectionner un processus avec la priorité la plus élevée dans l'une des files d'attente non vides ;
 \* Élire ce processus comme nouveau processus en cours d'exécution ;
 • Sinon si un processus est en cours d'exécution :
 \* Si le processus a terminé son exécution, le retirer du processeur ;
 \* Sinon,
 + incrémenter le temps d'utilisation du processus ;
 + Si des processus de priorité supérieure ou égale attendent :
 - Retirer le processus en cours d'exécution du processeur ;
 - Réduire sa priorité de 1 et le mettre dans la file d'attente correspondant à sa priorité ;
 - Élire un processus dont la priorité est la plus élevée parmi les processus des files d'attente non vides;
 + Sinon, réduire sa priorité de 1 et continuer à exécuter le processus en cours d'exécution.

 3. Parmi les propositions suivantes, donner la structure la plus adaptée pour stocker les processus d’une même priorité :

- Proposition 1 : Liste
- Proposition 2 : File
- Proposition 3 : Pile

Pour représenter le processus, on utilise une classe Processus qui possède les variables d’instances PID (l’identifiant du processus), priorite (la priorité du processus), temps\_utilisation sur le CPU et le temps nécessaire à son exécution temps\_CPU.

1. Compléter le constructeur de la classe Processus :

1 class Processus:
4 ...(self, ..., priorite, temps\_CPU):
5 ... priorite = priorite
7 ... PID = ...
8 self.temps\_utilisation = 0
9 self.temps\_CPU = temps\_CPU

1. On considère les trois processus suivants :
P1 = Processus(PID=1,priorite=0,temps\_CPU=10)
P2 = Processus(PID=2,priorite=0,temps\_CPU=7)
P3 = Processus(PID=3,priorite=0,temps\_CPU=5)

On a donc liste\_files=[[P3, P2, P1], [], []].

Compléter la simulation suivante, dans laquelle la variable CPU contient le processus en cours d’exécution :

 Cycle 1: CPU=P1 liste\_files=[[P3, P2], [], []]

 Cycle 2: CPU=P2 liste\_files=[[P3],[P1],[]]

 Cycle 3: CPU=P3 liste\_files=[[], [...],[]]

 Cycle 4: CPU=P3 liste\_files=[[], [...], [...]]

 Cycle 5: CPU=... liste\_files=[[], [...], [...]]

Dans les questions 6 et 7, on dispose :

* d’un processus qui nécessite un temps d’utilisation de 1000 pour terminer ;
* d’un nombre important de processus dont le temps d’utilisation pour terminer est de 4 où l’on suppose de plus que chaque processus terminé est remplacé par un nouveau processus similaire.
1. Expliquer pourquoi le processus qui nécessite un long temps d’utilisation du CPU risque de ne jamais terminer avec le programme de gestion de l’ordre des processus ci-dessus (indiquer notamment la priorité du processus long au bout de quelques temps).

Pour régler ce phénomène, on décide d’ajouter la variable d’instance temps\_d\_attente au processus, et on définit une constante appelée Max\_Temps qui correspond au temps maximum qu’un processus attend avant de remonter sa priorité.
L’idée est qu’à chaque cycle, le temps\_d\_attente augmente. Ainsi, si sa valeur dépasse Max\_Temps, alors sa priorité augmente.

1. Expliquer pourquoi le processus qui nécessite un temps long d’utilisation du CPU ne risque plus de ne jamais terminer avec ce nouveau programme de gestion de l’ordre des processus.
2. Écrire une fonction meilleur\_priorite qui renvoie None s’il n’y a plus de processus et la priorité de l’un des processus les plus prioritaires de la liste des files d’attente dans le cas contraire.

1 def meilleur\_priorite(liste\_files):
2 ...

Exemple :

# p1, p2 et p3 sont des instances de la classe 'Processus'
>>> liste\_files = [[], [p2], [p3, p1]]
>>> meilleur\_priorite(liste\_files)
1

1. Écrire une fonction prioritaire qui renvoie None si aucune des files d’attente de la liste ne contient un processus et qui renvoie l’un des processus parmi les plus prioritaires sinon (dans ce cas la fonction prioritaire supprimera le processus choisi de la file d’attente dans laquelle il se trouvait).

1 def prioritaire(liste\_files):
2 ...

On pourra utiliser liste.pop(i) pour renvoyer l’élément de la liste à la position i, tout en le supprimant de la liste.

1. Écrire une fonction gerer qui récupère le processus en cours d’exécution p ainsi que la liste des files d’attente liste\_files et qui implémente le programme donné en début d’énoncé pour gérer les processus.

1 def gerer(p, liste\_files):
2 ...

# EXERCICE 3 (8 points)

*Cet exercice porte sur la programmation Python (dictionnaire, récursivité, spécification), la programmation orientée objet, les bases de données relationnelles, les requêtes SQL et les arbres binaires.*

*Cet exercice est composé de 3 parties indépendantes.*

## Partie A

Dans cette partie, on s’intéresse à la gestion de la base de données d’un hôpital. On pourra utiliser les mots-clés SQL suivants : AND, FROM, INSERT, INTO, JOIN, ON, SELECT, SET, UPDATE, VALUES, WHERE. On utilisera également la fonction d’agrégation COUNT qui renvoie le nombre d’enregistrements correspondant à une requête.

La table Patient possède les attributs suivants :

* nom\_patient de type TEXT (clé primaire) ;
* prenom de type TEXT ;
* numero\_secu de type INT ;
* age de type INT.

| Patient |
| --- |
| nom\_patient | prenom | numero\_secu | age |
| Heartman | Alice | 207053523800187 | 17 |
| Douglas | Bob | 100017500155572 | 24 |
| Woods | Caroll | 258125930610747 | 65 |

La table Symptome possède les attributs suivants :

* nom\_patient de type TEXT (clé primaire et clé étrangère) ;
* toux de type TEXT ;
* fievre de type TEXT ;
* nausee de type TEXT ;
* anosmie de type TEXT.

| Symptome |
| --- |
| nom\_patient | toux | fievre | nausee | anosmie |
| Heartman | Oui | Non | Non | Oui |
| Douglas | Non | Oui | Oui | Non |
| Woods | Oui | Oui | Non | Non |

La table Maladie possède, entre autres, l’attribut nom\_maladie de type TEXT, qui est la clé primaire. Les autres attributs de cette table ne sont pas représentés car ils ne sont pas utiles pour l’exercice.

| Maladie |
| --- |
| nom\_maladie |
| Covid-19 |
| Gastroentérite |

La table Diagnostic possède les attributs suivants :

* nom\_patient de type TEXT (clé primaire et clé étrangère) ;
* nom\_maladie de type TEXT (clé étrangère).

| Diagnostic |
| --- |
| nom\_patient | nom\_maladie |
| …….. …….. …….. | …….. …….. …….. |

1. Écrire une requête SQL permettant d’obtenir les noms et prénoms des patients ayant strictement plus de 60 ans.
2. Alice Heartman ne tousse plus. Écrire une requête SQL permettant de mettre à jour la base de données avec cette information.
3. On souhaite effectuer des statistiques sur les symptômes des patients atteints de Covid-19. Écrire une requête SQL permettant de connaître le nombre de patients avec un diagnostic de Covid-19 qui toussent.

Un employé de l’hôpital saisit la requête suivante :

INSERT INTO Patients VALUES ('Douglas', 'Patrick', 168077230253829, 55)

1. Expliquer pourquoi cette requête produit une erreur.
2. Proposer une modification du schéma relationnel qui permettrait de résoudre ce problème.

## Partie B

On s’intéresse maintenant à l’automatisation du diagnostic à partir des symptômes. Cette automatisation se fait à l’aide d’un arbre de décision binaire, tel que celui illustré sur la figure 1.



Figure 1. Exemple d’arbre de décision binaire

Chaque nœud interne de l’arbre est étiqueté par un symptôme, et chaque feuille est étiquetée par un diagnostic. Pour établir un diagnostic, on se place à la racine et on parcourt l’arbre de la manière suivante :

* si on arrive à une feuille, le diagnostic est l’étiquette de cette feuille ;
* sinon, on regarde si l’étiquette du nœud est un des symptômes du patient. Si oui, on continue le parcours dans le sous-arbre droit, sinon, on continue le parcours dans le sous-arbre gauche.

L’arbre de la figure 1 donne un diagnostic pour la Covid-19. Par exemple, un patient qui ne tousse pas et n’a pas d’anosmie, mais a de la fièvre et des nausées est diagnostiqué négatif si on suit cet arbre de décision.

1. Donner le diagnostic pour un patient qui tousse et qui a de la fièvre, mais n’a pas de nausée ni d’anosmie, d’après l’arbre de la figure 1.

On décide d’implémenter les arbres binaires à l’aide de la classe Noeud ci-dessous :

1 class Noeud:
2 def \_\_init\_\_(self, valeur, gauche = None, droit = None):
3 """valeur correspond au symptome si le noeud est
4 interne ou au diagnostic si le noeud est une feuille"""
5 self.valeur = valeur
6 self.gauche = gauche
7 self.droit = droit
8
9 def est\_feuille(self):
10 """renvoie vrai si le noeud est une feuille faux sinon"""
11 return self.gauche == None and self.droit == None
12
13 def symptome(self):
14 assert not self.est\_feuille()
15 return self.valeur
16
17 def diagnostic(self):
18 assert self.est\_feuille()
19 return self.valeur

1. Préciser la signification de l’assertion de la méthode symptome.
2. Nommer un attribut et une méthode de la classe Noeud.

On représente les symptômes d’un patient en Python par un dictionnaire dont les clés sont les symptômes possibles, et les valeurs sont True si le patient présente ce symptôme et False sinon.

Par exemple, les symptômes du patient de la question 7 sont représentés par le dictionnaire suivant :

patient = {'toux' : True, 'fievre' : True, 'nausee' : False,
 'anosmie' : False}

1. Compléter la fonction applique suivante, définie récursivement, qui renvoie le diagnostic établi en utilisant un arbre de décision binaire implémenté à l’aide de la classe Noeud précédente.

def applique(arbre, patient):
 if arbre.est\_feuille():
 ...
 else:
 if patient[arbre.symptome()]:
 ...
 else:
 ...

1. Donner la taille de l’arbre représenté en figure 1. On considère que la taille d’un arbre constitué d’une unique feuille est 1.

On souhaite réduire la taille de cet arbre en utilisant l’observation suivante :
un nœud dont les deux sous-arbres sont des feuilles correspondant au même diagnostic peut être remplacé par une feuille correspondant à ce diagnostic, comme illustré en figure 2.



Figure 2. Règle de réduction pour les arbres de décision binaire

1. Appliquer cette règle à l’arbre de la figure 1 pour le réduire et dessiner le nouvel arbre.
2. Compléter la méthode reduire qui permet d’appliquer cette règle récursivement pour réduire la taille d’un arbre de décision binaire.

1 def reduire(self):
2 """fonction récursive qui réduit la taille d'un arbre de
3 décision sans changer les décisions prises"""
4 if self.est\_feuille():
5 return
6 self.gauche.reduire()
7 self. ...
8 if self.gauche.est\_feuille() and ... \
9 and ... == ... :
10 self.valeur = ...
11 self.gauche = ...
12 self.droite = ...

## Partie C

Dans cette partie, on s’intéresse à l’intégrité et à la sécurité des données.

Sur les 15 chiffres du numéro de sécurité sociale, 2 servent à détecter les erreurs :
étant donné le nombre n formé des 13 premiers chiffres, le nombre k formé des 2 derniers chiffres, appelé la clé, est choisi pour que n$+$k soit un multiple de 97.

Par exemple, 207053523800187 est bien formé car :

2070535238001 + 87 = 97 x 21345724104.

On rappelle que les opérateurs % et // permettent en Python d’obtenir respectivement le reste et le quotient dans une division euclidienne. Par exemple : 13 % 3 renvoie 1 et 13//3 renvoie 4 (car $13=3×4+1$). On peut donc vérifier qu’un nombre entier $n$ est un multiple de $p$ en testant si le reste de la division de $n$ par $p$ vaut zéro.

1. Recopier et compléter la fonction verifie suivante qui renvoie un booléen indiquant si un numéro de sécurité sociale représenté par un entier (type int) est bien formé.

1 def verifie(num\_secu):
2 n = num\_secu // 100
3 k = num\_secu % 100
4 return ...

1. Compléter la fonction cle qui permet de renvoyer la clé k d’un numéro de sécurité sociale en prenant pour paramètre le nombre n formé des 13 premiers chiffres du numéro de sécurité sociale.

1 def cle(n):
2 ...