

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

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 8 pages numérotées de 1/8 à 8/8.

Le candidat traite les 3 exercices proposés.

EXERCICE 1 (6 points)

Cet exercice porte sur la programmation orientée objet et les dictionnaires.

Dans le tableau ci-dessous, on donne les caractéristiques nutritionnelles, pour une quantité de 100 grammes, de quelques aliments.

	Lait entier UHT	Farine de blé	Huile de tournesol
Énergie (kcal)	65.1	343	900
Protéines (grammes)	3.32	11.7	0
Glucides (grammes)	4.85	69.3	0
Lipides (grammes)	3.63	0.8	100

Figure 1 : Caractéristiques nutritionnelles

Pour chaque aliment, on souhaite stocker les informations dans un objet de la classe `Aliment` définie ci-dessous, où `e`, `p`, `g` et `l` sont de type `float` et désignent respectivement les quantités d'énergie, de protéines, de glucides et de lipides de l'aliment.

```
1. class Aliment:
2.     def __init__(self, e, p, g, l):
3.         self.energie = e
4.         self.proteines = p
5.         self.glucides = g
6.         self.lipides = l
```

1.

- Ecrire, à l'aide du tableau des caractéristiques nutritionnelles de la Figure 1, l'instruction en langage Python pour instancier l'objet `lait`.
- Donner l'instruction qui permet d'obtenir la valeur 65.1 de l'objet `lait` instancié dans la question précédente.

Une erreur s'est introduite dans le tableau de la Figure 1 : la masse de protéines dans le lait est 3.4 au lieu de 3.32.

- Donner l'instruction qui modifie la masse de protéines de l'objet `lait` instancié dans la question 1.a.

On souhaite ajouter une méthode "`energie_reelle`" à la classe `Aliment` qui calcule l'énergie en kcal d'un aliment en fonction d'une masse donnée.

Par exemple :

Pour 245 grammes de lait, l'énergie réelle sera $245 \times 65.1 \div 100 = 159.495$ kcal.

L'instruction `lait.energie_reelle(245)` renvoie alors 159.495

2. Recopier et compléter les lignes n°1 et n°2 dans la méthode ci-dessous.

1. def energie_reelle(.....,masse):
2. return

3.

On regroupe les caractéristiques nutritionnelles du tableau de la Figure 1 dans le dictionnaire suivant, les clés étant des chaînes de caractères donnant le nom de l'aliment et les valeurs associées des objets de la classe `Aliment` :

```
nutrition = {'lait' : Aliment(65.1,3.4,4.85,3.63),
             'farine' : Aliment(343,11.7,69.3,0.8),
             'huile' : Aliment(900,0,0,100)
            }
```

- a. Donner l'instruction qui permet d'obtenir la valeur énergétique en kcal du lait à partir des données de ce dictionnaire.
- b. Donner l'instruction qui permet d'obtenir la valeur énergétique réelle de 220 grammes de lait à partir des données de ce dictionnaire.

Une recette de gâteau (sans œuf) utilise les ingrédients suivants :

- 230 g de farine ;
- 220 g de lait ;
- 100 g d'huile.

Les quantités d'ingrédients, en grammes, sont regroupées dans le dictionnaire suivant :

```
recette_gateau={'lait' : 220, 'farine' :230, 'huile':100}
```

4.

Ecrire, en utilisant la classe `Aliment` et la méthode `energie_reelle`, les instructions nécessaires pour calculer l'énergie réelle totale du gâteau.

EXERCICE 2 (6 points)

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

L'énoncé de cet exercice utilise les mots-clefs du langage SQL suivants : SELECT, FROM, WHERE, JOIN...ON, UPDATE...SET, INSERT INTO...VALUES.

Un orchestre souhaite créer une base de données relationnelle contenant la table de ses membres et celle des instruments joués. Pour cela, il est choisi d'utiliser le langage SQL. Une relation `musiciens` est créée. Elle contient également les informations d'inscription ainsi que leur ancienneté au sein de l'orchestre.

`id_mus` est la clé primaire de la relation `musiciens`.

`Code_instrument` est la clé primaire de la relation `instruments`.

<u>id_mus</u>	Nom	Prenom	Mail	Anciennete	Code_instrument
1	Dupont	Claudine	cDupont@mail.fr	5	1
2	Leclerc	Jean	leclerc.jean@mail.fr	2	2
3	M'bake	Doume	doubake@mail.fr	7	5
4	Descarte	Hugo	Hugo.arnaque@mail.fr	4	1
5	Tourelle	Charlène	char.tour@mail.fr	1	7
6	Dupuis	Alice	dupuis.Alice@mail.fr	9	4
7	Tourelle	Phillipe	filoutou@mail.fr	1	4
8	Fernandez	Kader	kader454@mail.fr	4	1
9	Dujardin	Laure	dujardin.laure@mail.fr	1	6
10	Lefevre	Marie	lefevre.marie@mail.fr	2	2

Extrait de la relation `musiciens`

Afin de compléter les informations personnelles, la relation `instruments` est créée :

<u>Code_instrument</u>	type_ins
1	Violon
2	Violon
3	Alto
4	Alto
5	Violoncelle
6	Contrebasse
7	Harpe

Extrait de la relation `instruments`

1.

- Donner la clé étrangère de la relation `musiciens` en justifiant la réponse.
- Ecrire le schéma relationnel des relations `musiciens` et `instruments`.

2.

a. Écrire le résultat de la requête suivante :

```
SELECT Nom, Prenom FROM musiciens WHERE  
Code_instrument=4;
```

b. Ecrire une requête permettant d'afficher les noms et prénoms de tous les musiciens membres depuis 5 ans ou plus.

3.

a. Donner la requête permettant de modifier l'adresse mail de Tourelle Charlène par tour.char@mail.fr.

b. Ecrire la requête permettant d'afficher le nom et prénom de tous les musiciens violonistes à l'aide d'une jointure.

4. Afin de gérer la partie administrative, une nouvelle relation `Admin` est créée. Elle contient un intitulé du poste en clé primaire, l'identifiant du musicien associé et son ancienneté dans le poste.

<u>Poste</u>	<u>id_mus</u>	<u>Anciennete_poste</u>
Archiviste	8	1
Chef d'Orchestre	4	4
Président	6	7

Extrait de la relation `Admin`

a. Insérer le poste Trésorier qui vient d'être créé par l'orchestre et dont le rôle vient d'être attribué à Leclerc Jean dans la table `Admin`.

b. Ecrire la requête permettant d'afficher l'instrument joué par le Président, en utilisant des jointures.

EXERCICE 3 (8 points)

Cet exercice porte sur les arbres binaires et la POO.

```
class ArbreBinaire:
    """ Construit un arbre binaire """

    def __init__(self, valeur):
        """ Crée une instance correspondant
        à un état initial """
        self.valeur = valeur
        self.enfant_gauche = None
        self.enfant_droit = None

    def insert_gauche(self, valeur):
        """ Insère le paramètre valeur
        comme fils gauche """
        if self.enfant_gauche is None:
            self.enfant_gauche = ArbreBinaire(valeur)
        else:
            new_node = ArbreBinaire(valeur)
            new_node.enfant_gauche = self.enfant_gauche
            self.enfant_gauche = new_node

    def insert_droit(self, valeur):
        """ Insère le paramètre valeur
        comme fils droit """
        if self.enfant_droit is None:
            self.enfant_droit = ArbreBinaire(valeur)
        else:
            new_node = ArbreBinaire(valeur)
            new_node.enfant_droit = self.enfant_droit
            self.enfant_droit = new_node

    def get_valeur(self):
        """ Renvoie la valeur de la racine """
        return self.valeur

    def get_gauche(self):
        """ Renvoie le sous arbre gauche """
        return self.enfant_gauche

    def get_droit(self):
        """ Renvoie le sous arbre droit """
        return self.enfant_droit
```

- a. En utilisant la classe définie ci-dessus, donner un exemple d'attribut, puis un exemple de méthode.

- b. Après avoir défini la classe `ArbreBinaire`, on exécute les instructions Python suivantes :

```
r = ArbreBinaire(15)
r.insert_gauche(6)
r.insert_droit(18)
a = r.get_valeur()
b = r.get_gauche()
c = b.get_valeur()
```

Donner les valeurs associées aux variables `a` et `c` après l'exécution de ce code.

On utilise maintenant la classe `ArbreBinaire` pour implémenter un arbre binaire de recherche.

On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel :

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de `x`, alors il faut que `y.valeur <= x.valeur`.
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur > x.valeur`.

2. On exécute le code Python suivant. Représenter graphiquement l'arbre ainsi obtenu.

```
racine_r = ArbreBinaire(15)
racine_r.insert_gauche(6)
racine_r.insert_droit(18)

r_6 = racine_r.get_gauche()
r_6.insert_gauche(3)
r_6.insert_droit(7)

r_18 = racine_r.get_droit()
r_18.insert_gauche(17)
r_18.insert_droit(20)

r_3 = r_6.get_gauche()
r_3.insert_gauche(2)
```

3. On a représenté sur la figure 1 ci-dessous un arbre. Justifier qu'il ne s'agit pas d'un arbre binaire de recherche. Redessiner cet arbre sur votre copie en conservant l'ensemble des valeurs `{2,3,5,10,11,12,13}` pour les nœuds afin qu'il devienne un arbre binaire de recherche.

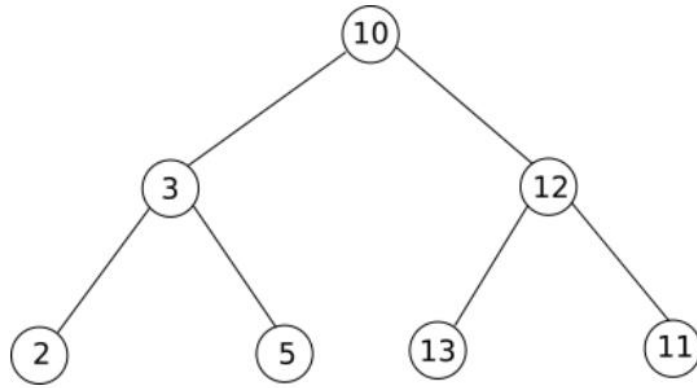


Figure 1

4. On considère qu'on a implémenté un objet `ArbreBinaire` nommé `A` représenté sur la figure 2.

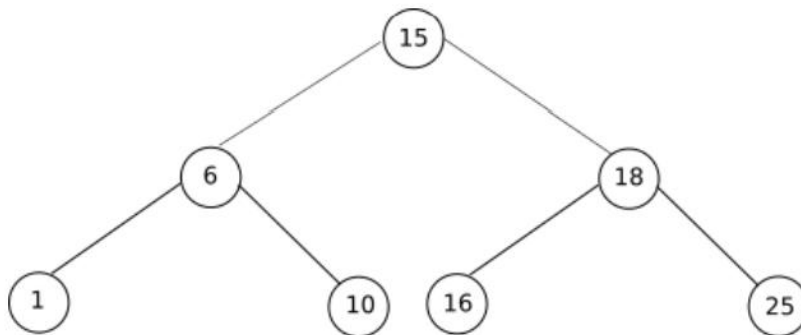


Figure 2

On définit la fonction `parcours_infixe` suivante, qui prend en paramètre un objet `ArbreBinaire` `T` et un second paramètre `parcours` de type liste.

Numéro de lignes	Fonction <code>parcours_infixe</code>
1	<code>def parcours_infixe(T, parcours):</code>
2	<code> """ Affiche la liste des valeurs de l'arbre """</code>
3	<code> if T is not None:</code>
4	<code> parcours_infixe(T.get_gauche(), parcours)</code>
5	<code> parcours.append(T.get_valeur())</code>
6	<code> parcours_infixe(T.get_droit(), parcours)</code>
7	<code> return parcours</code>

Donner la liste renvoyée par l'appel suivant : `parcours_infixe(A, [])`.