Devoir sur table

Durée de l'épreuve : 1h
L'usage de la calculatrice n'est pas autorisé.

Exercice 1

Faire correspondre chaque type de test à sa définition.

	Les tests
a	Unitaires
b	D'intégration
c	Fonctionnels (boîte noire)
d	Structurels (boîte blanche)
e	De performance
f	D'utilisabilité

	Permettent de valider
1	Plusieurs parties du programme ensemble
2	Par rapport à la spécification
3	Une partie du programme
4	L'ergonomie du programme
5	Par rapport à la taille des données
6	Par rapport à l'implémentation

Réponses : a-3 b-1 c-2 d-6 e-5 f-4

Exercice 2

Questions à choix multiple sur la Programmation Orientée Objet.

Question 1 :
Quelle est la méthode spéciale utilisée pour initialiser un objet dans une classe Python ?
Réponse : B. __init__
Explication : La méthode spéciale __init__ est appelée automatiquement lorsqu'un objet est créé à partir d'une classe. Elle est utilisée pour initialiser les attributs de l'objet.

Question 2 :
En Python, comment appelle-t-on une méthode d'instance depuis un objet ?
Réponse : B. object.method_name()
Explication : Une méthode d'instance est appelée en utilisant un objet (instance de la

classe) suivi de la notation pointée : objet.nom_methode(). Le mot-clé self est utilisé à l'intérieur de la méthode pour référencer l'objet, mais il n'est pas explicitement passé lors de l'appel.

Question 3 :
Dans une classe Python, que représente l'attribut self ?
Réponse : C. L'objet courant de la classe
Explication : Le mot-clé self représente l'instance actuelle de la classe. Il est utilisé pour accéder aux attributs et méthodes de l'objet courant à l'intérieur de la classe.

Question 4 :
Quelle est l'utilité des méthodes statiques dans une classe Python ?
Réponse : A. Elles ne nécessitent pas de paramètre self et peuvent être appelées sans créer d'instance
Explication : Une méthode statique appartient à la classe, mais elle n'a pas besoin de référence à une instance (self). On peut l'appeler directement à partir de la classe ou d'une instance.

Exercice 3

```
class Rectangle:
  def __init__(self, largeur, hauteur):
    """Initialise un rectangle avec sa largeur et sa hauteur."""
    self.largeur = largeur
    self.hauteur = hauteur

  def surface(self):
    """Retourne la surface du rectangle."""
    return self.largeur * self.hauteur

  def __eq__(self, other):
    """Surcharge de l'opérateur == (égalité)."""
    return self.surface() == other.surface()


  def __lt__(self, other):
    """Surcharge de l'opérateur < (inférieur)."""
    return self.surface() < other.surface()


  def __gt__(self, other):
    """Surcharge de l'opérateur > (supérieur)."""
    return self.surface() > other.surface()

  def __len__(self):
    """Surcharge de len() pour retourner la surface du rectangle."""
```

```
        return self.surface()


r1 = Rectangle(4, 5)  # Surface : 20
r2 = Rectangle(3, 7)  # Surface : 21
r3 = Rectangle(2, 10) # Surface : 20

print(r1 == r2)  # False, car 20 != 21
print(r1 < r2)   # True, car 20 < 21
print(r3 > r2)   # False, car 20 > 21 est faux
print(len(r1))   # 20, la surface du rectangle r1
```

Exercice 4

```
class Eleve:
    def __init__(self, nom, prenom, moyenne):
        """Initialise un élève avec son nom, prénom et moyenne."""
        self.nom = nom
        self.prenom = prenom
        self.moyenne = moyenne

    def __str__(self):
        """Affiche les informations de l'élève sous une forme lisible."""
        return f"{self.prenom} {self.nom} - Moyenne : {self.moyenne:.1f}"


class Classe:
    def __init__(self):
        """Initialise une classe vide (sans élèves)."""
        self.eleves = []

    def ajouter_eleve(self, eleve):
        """Ajoute un élève à la liste des élèves."""
        if isinstance(eleve, Eleve):
            self.eleves.append(eleve)
        else:
            print("Erreur : vous devez ajouter un objet de type Eleve.")

    def afficher_classe(self):
        """Affiche tous les élèves de la classe."""
        if not self.eleves:
            print("La classe est vide.")
        else:
            for eleve in self.eleves:
                print(eleve)

    def moyenne_classe(self):
        """Calcule et retourne la moyenne générale de la classe."""
        if not self.eleves:
            return 0.0  # Aucun élève, donc moyenne de 0
        total_moyennes = 0
        for eleve in self.eleves:
            total_moyennes += eleve.moyenne
        return total_moyennes / len(self.eleves)
```

Exercice 5

```
class Pile:

  def __init__(self):
    self.contenu = []
  def est_vide(self):

    return self.contenu == []
  def empiler(self, v):

    self.contenu.append(v)
  def depiler(self):

    assert not self.est_vide()
    return self.contenu.pop()

def eval_expression(tab):
  p = Pile()
  for element in tab:
    if element != '+' and element != '*':
      p.empiler(element)
    else:
      if element == '+':
        resultat = p.depiler() + p.depiler()
      else:
        resultat = p.depiler() * p.depiler()
      p.empiler(resultat)
  return p.depiler()
```

Exercices 6

*Quelle opération n'est **pas possible** dans une pile (structure LIFO) ?*
Réponse : C. Accéder directement à l'élément du bas de la pile
Explication : Une pile suit le principe LIFO (Last In, First Out).

Quelle est la principale différence entre une file et une pile ?
Réponse : B. La file suit un ordre FIFO (First In First Out), alors que la pile suit un ordre LIFO (Last In First Out)
Explication :

- Une **file** traite les éléments dans l'ordre d'arrivée (FIFO).
- Une **pile** traite les éléments en dernier arrivé, premier sorti (LIFO).

Dans une liste chaînée, que se passe-t-il lorsqu'on supprime un nœud situé au milieu de la liste ?
Réponse : B. Le pointeur du nœud précédent est mis à jour pour pointer vers le nœud suivant
Explication : Lorsqu'un nœud est supprimé, la structure de la liste chaînée reste intacte en ajustant le pointeur du nœud précédent pour pointer directement vers le nœud suivant.

Exercice 7

```
class Cellule:
    def __init__(self, valeur, suivante):
        self.valeur = valeur
        self.suivante = suivante

class ListeChainee:
    def __init__(self):
        self.tete = None

    def est_vide(self):
        return self.tete is None

    def taille(self):
        """Calcule la taille de la liste."""
        courant = self.tete
        compteur = 0
        while courant:
            compteur += 1
            courant = courant.suivante
        return compteur

    def chercher_element(self, valeur):
        """Vérifie si un élément est présent dans la liste."""
        courant = self.tete
        while courant:
            if courant.valeur == valeur:
                return True
            courant = courant.suivante
        return False

    def ajouter_debut(self, valeur):
        """Ajoute une cellule au début de la liste."""
        nouvelle_cellule = Cellule(valeur, self.tete)
        self.tete = nouvelle_cellule
```