



Math93.com

NSI - DS 1 - Tle (Correction) POO - Les classes

Septembre 2025 - Durée 66 min (100 min si accommodation)

Partie 1 : épreuve écrite

Exercice 1. Une Bibliothèque

Thème abordé : programmation objet en langage Python

On s'intéresse à la gestion d'une bibliothèque. Une bibliothèque possède des livres et des lecteurs qui peuvent emprunter des livres. On souhaite modéliser ce système en utilisant la programmation orientée objet en Python.

Partie 1 : Classe Livre

On considère la classe suivante :

```
class Livre:
    """Modélise un livre de la bibliothèque"""
    def __init__(self, titre, auteur):
        """
        Initialise un livre
        :param titre (str) le titre du livre
        :param auteur (str) l'auteur du livre
        """
        self.titre = titre
        self.auteur = auteur
        self.disponible = True
```

1. Déterminer les types de : Livre, titre, auteur, disponible et __init__ (classe, attribut, méthode ou objet ...).



Corrigé (2 pts)

- titre : attribut
- auteur : attribut
- disponible : attribut
- __init__ : méthode (constructeur)
- La classe est Livre, une instance est un objet.

2. Indiquer le type attendu pour le paramètre titre.



Corrigé (0.5 pt)

- | str (chaîne de caractères).

3. Comment définir la variable self .

**Corrigé (0.5 pt)**

| La variable `self` désigne une instance de la classe `Livre`.

4. Écrire une instruction permettant de créer une instance `livre1` représentant «Le Petit Prince» d'Antoine de Saint-Exupéry.

**Corrigé (1 pt)**

| On appelle le constructeur de `Livre` avec le titre et l'auteur.

```
livre1 = Livre("Le Petit Prince", "Antoine de Saint-Exupéry")
```

5. Compléter la méthode suivante :

```
def est_disponible(self):
    """
    Renvoie True si le livre est disponible, False sinon.
    """
    ...
```

**Corrigé (3 pts)**

| La méthode renvoie la valeur booléenne stockée dans l'attribut `disponible`.

```
def est_disponible(self):
    return self.disponible
```

Partie 2 : Classe Lecteur

On modélise maintenant un lecteur :

```
class Lecteur:
    """Modélise un lecteur de la bibliothèque"""
    def __init__(self, nom):
        self.nom = nom
        self.emprunts = [] # liste des livres empruntés
```

5. Compléter la méthode emprunter :

```
def emprunter(self, livre):
    """
    Ajoute le livre à la liste des emprunts si le livre est disponible,
    et marque le livre comme non disponible.
    Renvoie True si l'emprunt réussit, False sinon.
    """
    ...
```



Corrigé (3 pts)

Vérifier `livre.disponible`. Si True, ajouter à `self.emprunts`, passer `livre.disponible` à False, et renvoyer True. Sinon renvoyer False.

```
def emprunter(self, livre):
    if livre.disponible:
        self.emprunts.append(livre)
        livre.disponible = False
        return True
    return False
```

6. Compléter la méthode rendre :

```
def rendre(self, livre):
    """
    Retire le livre de la liste des emprunts et marque le livre
    comme disponible.
    Renvoie True si la restitution réussit, False sinon.
    """
    ...
```



Corrigé (2 pts)

Ne rendre que si `livre` est dans `self.emprunts` : le retirer, mettre `livre.disponible = True`, renvoyer True. Sinon False.

```
def rendre(self, livre):
    if livre in self.emprunts:
        self.emprunts.remove(livre)
        livre.disponible = True
    return True
    return False
```

Partie 3 : Classe Bibliotheque

Une bibliothèque contient plusieurs livres :

```
class Bibliotheque:
    """Modélise une bibliothèque"""
    def __init__(self, nom):
        self.nom = nom
        self.livres = [] # liste d'objets Livre
```

7. Compléter la méthode ajouter_livre :

```
def ajouter_livre(self, livre):
    """
    Ajoute le livre à la bibliothèque.
    """
    ...
```



Corrigé (1 pt)

| Ajouter l'objet livre à la liste self.livres.

```
def ajouter_livre(self, livre):
    self.livres.append(livre)
```

8. Compléter la méthode livres_disponibles :

```
def livres_disponibles(self):
    """
    Renvoie la liste des titres des livres encore disponibles.
    """
    ...
```



Corrigé (2 pts)

| Filtrer les livres dont disponible vaut True et renvoyer leurs titres.

```
def livres_disponibles(self):
    return [livre.titre for livre in self.livres if livre.disponible]
```

9. Compléter la méthode chercher_auteur :

```
def chercher_auteur(self, auteur):
    """
    Renvoie la liste des titres de livres écrits par l'auteur
    passé en paramètre.
    """
    ...
```



Corrigé (3 pts)

| Parcourir self.livres et sélectionner les titres dont livre.auteur == auteur.

```
def chercher_auteur(self, auteur):
    return [livre.titre for livre in self.livres
            if livre.auteur == auteur]
```

Partie 4 : Algorithmique

10. On souhaite savoir quel lecteur a emprunté le plus grand nombre de livres. On dispose d'une liste tab_lecteurs d'instances de Lecteur. Écrire la fonction lecteur_max :

```
def lecteur_max(tab_lecteurs):
    """
    Renvoie le lecteur ayant emprunté le plus de livres.
    Si plusieurs lecteurs ont le même maximum, renvoie l'un d'entre eux.
    Si la liste est vide, renvoie None.
    """
    ...
```

**Corrigé (3 pts)**

Parcourir la liste, mémoriser le meilleur vu jusqu'ici (nombre d'emprunts maximal). Gérer le cas de la liste vide en renvoyant None.

```
def lecteur_max(tab_lecteurs):  
    if not tab_lecteurs:  
        return None  
    max_lecteur = tab_lecteurs[0]  
    max_n = len(tab_lecteurs[0].emprunts)  
    for lec in tab_lecteurs[1:]:  
        n = len(lec.emprunts)  
        if n > max_n:  
            max_n = n  
            max_lecteur = lec  
    return max_lecteur
```

↔ **Fin du devoir** ↔

**Remarque**

Rendez votre copie avec le sujet au professeur puis travailler sur votre TD.