Examen final : Compléments d'informatique - Correction

ENSAI 2A UE3

2025-09-29

- Examen sur papier
- Durée : 2h
- Une feuille A4 *manuscrite* recto/verso autorisée
- Sans calculatrice

> ! À lire avant de commencer
>
> Le sujet est composé de plusieurs pages, numérotées de 1 à plusieurs pages.
>
> - Les exercices peuvent être traités dans l'ordre de votre choix.
> - Dans chaque exercice, les questions ne sont pas forcément de difficulté croissante.
> - Sauf mention contraire, les différentes questions attendent des réponses courtes.
> - Respectez scrupuleusement les consignes. Cependant, si une question ne vous semble pas claire, notez sur votre copie la manière dont vous l'avez comprise et traitée.
> - Quand du code est demandé, veuillez respecter les règles de base du langage (mots clefs et indentation). Des erreurs seront tolérées, car vous êtes sur papier. En cas d'oubli d'un nom de méthode Python, utilisez du pseudo-code.
> - Excepté si c'est demandé, il n'est pas nécessaire d'écrire la documentation du code (même si la doc c'est bien !)

Bonne chance !

1 Questions de cours (7 points)

a. **Citez les principaux outils et logiciels utilisés pendant les TP.**

- Visual Studio Code, python, PostgreSQL, Git, GitHub, cloudBeaver…

b. **Pourquoi écrire du code de qualité ?**

- Maintenabilité : plus facile à comprendre, à maintenir et à modifier
- Lisibilité : facilite la collaboration et réduit les erreurs
- Performance : fonctionne mieux et consomme moins de ressources
- Sécurité : moins susceptible de contenir des vulnérabilités

c. **Quels sont les intérêts de découper votre application en couches ?**

- Travail en groupe
- Lisibilité du code
- Débogages

d. **Comment définiriez-vous les relations entre ces quatre classes : Velo, DeuxRoues, Trottinette, VeloElectrique**

- DeuxRoues : classe mère abstraite qui représente le concept général d'un véhicule à deux roues.
- Velo, Trottinette : héritent de la classe DeuxRoues
- VeloElectrique : hérite de Velo, c'est un Velo avec des attributs ou méthodes supplémentaires

e. **Qu'est-ce qu'un design pattern ?**

- C'est une bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels

f. **Quels principes doit respecter un bon test unitaire ?**

- Teste une seule fonctionnalité
- Isolé
- Reproductible
- Déterministe

g. **À quoi sert un mock ?**

- C'est un objet simulé qui remplace un composant réel lors des tests. Il permet de tester une unité de code en isolant ses dépendances externes.

h. **Relevez les erreurs et les mauvaises pratiques du code ci-dessous ? Quelles conséquences cela pourrait-il avoir ?**

```
from dao.db_connection import DBConnection
from business_object.joueuse import Joueuse

class JoueuseDao(metaclass=Singleton):
    def se_connecter(self, pseudo, mdp) -> Joueuse:
        with DBConnection().connection as connection:
            with connection.cursor() as cursor:
                requete = " SELECT *                          "
                requete += "  FROM joueuse                    "
```

```
            requete += " WHERE pseudo = '" + pseudo + "'    "
            requete += "   AND mdp = '" + mdp + "';         "
            cursor.execute(requete)
            res = cursor.fetchall()
    return res
```

- Possibilité d'injection SQL
  - Solution : Utiliser des requêtes paramétrées avec des placeholders comme `cursor.execute("SELECT * FROM joueuse WHERE pseudo = %s AND mdp = %s", (pseudo, mdp))`.
- Utilisation de `fetchall()` alors que l'on attend un seul résultat
  - Solution : Utiliser `fetchone()` à la place de `fetchall()`.
- Retour direct de `res` alors qu'on devrait générer un objet de type `Joueuse`
  - Solution : Mapper les résultats SQL à un objet `Joueuse`, par exemple : `j = Joueuse(nom=res["nom"], prenom=res["prenom"], pseudo=res["pseudo"] ...`.
- Pas de doc
- Les mots de passe sont potentiellement stockés en clair

2 Webservices (3 points)

a. **Nommez les différents éléments de la requête suivante :**

```
GET https://my-awesome-ws.fr/attack/8
```

- GET : Méthode HTTP
- https : Protocole
- my-awesome-ws.fr : host, adresse du webservice
- /attack/8 : endpoint

b. **Selon vous que pourrait faire cette requête ?**

```
PUT https://my-awesome-ws.fr/pokemon/15
```

Cette requête pourrait mettre à jour le pokemon d'id 15.

c. **Pour que cette requête fonctionne, que faut-il ajouter ?**

Pour faire une mise à jour, il faut fournir les éléments à mettre à jour dans un body au format json, par exemple :

```
{name: "Pikachu",
level: 5
...}
```

d. **Quelles sont les méthodes HTTP correspondant au CRUD ?**

- POST : Create
- GET : Read
- PUT : Update
- DELETE : Delete

3 Code (6 points)

Dans cet exercice, vous devez respecter les principes de la programmation en couche.

Vous allez créer une classe `Personne` avec les attributs suivants : nom, prenom, ville, age, est_adulte.

a. **Créez la classe et le constructeur. Chaque nouvelle personne créée a pour âge 0 et elle n'est pas adulte.**

```python
class Personne:
    def __init__(self, nom, prenom, ville):
        self.nom = nom
        self.prenom = prenom
        self.ville = ville
        self.age = 0
        self.est_adulte = False
```

b. **Ajoutez la méthode `anniversaire()` qui incrémente l'age. À 18 ans, la personne devient adulte. Écrivez la documentation de cette méthode.**

```python
    def anniversaire(self):
        """
        Incremente l age d une personne d une annee
        Si la personne atteint 18 ans, elle devient adulte.

        Retour :
            None
        """
        self.age += 1
        if self.age >= 18:
            self.est_adulte = True
```

c. **Écrivez, en utilisant `pytest`, les tests unitaires de la méthode `anniversaire()` ci-dessus.**

```python
from personne import Personne

class TestPersonne:

    def test_anniversaire_increment_age(self):
        # GIVEN
        p = Personne("Dupont", "Jean", "Paris")
        # WHEN
        p.anniversaire()
        # THEN
        assert p.age == 1
        assert not p.est_adulte

    def test_anniversaire_devient_adulte(self):
        # GIVEN
```

```
        p = Personne("Martin", "Claire", "Lyon")
        # WHEN
        for _ in range(18):
            p.anniversaire()
        # THEN
        assert p.age == 18
        assert p.est_adulte
```

d. **En supposant qu'il existe une table personne contenant les mêmes informations que la classe Personne, écrivez la requête SQL qui permet d'afficher le nombre de personnes regroupées par ville.**

```
SELECT ville,
       COUNT(*) AS nombre_personnes
  FROM personne
 GROUP BY ville;
```

e. **En utilisant la méthode `lister_toutes()` de la classe `PersonneDao` qui retourne la liste de toutes les personnes, créer une méthode de service qui retourne la liste de tous les adultes.**

```
from dao.personne_dao import PersonneDao

class PersonneService:

    def lister_adultes(self) -> list[Personne]:
        adultes = []
        personnes = PersonneDao().lister_toutes()
        for p in personnes:
            if p.est_adulte:
                adultes.append(p)
        return adultes
```

f. **Bonus : Modifiez la méthode de service afin qu'elle accepte un paramètre optionnel villes (une liste de villes). Si ce paramètre est fourni, la méthode doit retourner uniquement les adultes dont la ville figure dans cette liste. Si le paramètre n'est pas renseigné, la méthode doit retourner l'ensemble des adultes sans filtrage supplémentaire.**

```
class PersonneService:
    def lister_adultes(self, villes=None):
        adultes_villes=[]
        personnes = PersonneDao().lister_toutes()

        adultes = [p for p in personnes if p.est_adulte]
        if villes is not None:
            adultes_villes = [p for p in adultes if p.ville in villes]

        return adultes_villes
```

4 Git et Versionnage (4 points)

Votre cheffe de projet vous demande de mettre à jour le code du fichier `src/main.py` présent sur le dépôt git `https://github.com/soleil59/cultures-durables.git`.

La mise à jour consiste à se connecter à une base de données pour récupérer le contenu d'une table `chicon`. Vous n'avez jamais travaillé sur ce dépôt mais vous avez les droits nécessaires et un jeton opérationnel pour communiquer avec le dépôt distant.

a. **Comment procédez-vous pour récupérer le contenu du dépôt sur une machine locale ?**

Je crée un clone sur ma machine :

```
git clone https://github.com/soleil59/cultures-durables.git
```

ou mieux en intégrant le token :

```
git clone https://$TOKEN@github.com/soleil59/cultures-durables.git
```

Vous mettez ensuite à jour le fichier `src/main.py`. Comme vous codez proprement et que vous respectez les règles de sécurité, vous ne stockez pas "en dur" les identifiants de connexion mais dans un fichier que vous créez : `.env`.

b. **Comment procédez-vous pour envoyer votre code vers le dépôt distant ?**

- `git status` : vous commencez par lister les fichiers modifiés
- Si le fichier `.env` apparaît, il ne faut surtout pas l'envoyer vers le remote car il contient des informations de connexion. Ajoutez-le au fichier `.gitignore`
- Ensuite `git add .` pour ajouter les fichiers au prochain commit
- `git commit -m "maj chicon"`
- `git pull` (optionnel, mais devient obligatoire si le push ne fonctionne pas parce que votre dépôt local est en retard de version par rapport au remote)
- `git push`

Vous avez bien suivi la procédure, mais vous rencontrez un conflit.

c. **Après quelle commande peut-on avoir un conflit ? Expliquez brièvement.**

Un conflit peut survenir après la commande : `git pull`.

Un conflit apparaît lorsque Git ne peut pas automatiquement fusionner des modifications provenant de différentes sources, par exemple, si le même fichier ou la même ligne a été modifié à la fois en local et sur le remote. Dans ce cas, Git demande à l'utilisateur de résoudre manuellement le conflit avant de poursuivre.

d. **Proposez une hypothèse, sur la raison qui a pu provoquer un conflit.**

Une autre personne a modifié entre temps le fichier `src/main.py` et l'a envoyé au dépôt distant.

e. **Comment faites-vous pour le résoudre ?**

Vous ouvrez les fichiers et repérez les sections en conflit :

```
<<<<<<< HEAD
(votre version locale)
=======
(version distante)
>>>>>>> nom branche distante
```

Ensuite, vous décidez quelle version vous souhaitez conserver, puis refaites, add, commit et push.