Examen final : Remise à niveau informatique

ENSAI 2A UE3

2025-09-24

Instructions

- Paper-based exam
- Duration: 1h30
- No documents
- No calculator

> ! Read before starting
>
> The exam consists of several pages, numbered from 1 to the last page.
>
> - The exam may seem a little long. Do not waste time if you are stuck on a question; move on to the next one.
> - The exercises can be completed in any order.
> - Within each exercise, the questions are not necessarily arranged in increasing order of difficulty.
> - The different questions require short answers.
> - Follow the instructions carefully. However, if a question does not seem clear to you, write on your answer sheet how you understood and approached it.
> - When code is requested, please follow the basic rules of the language (keywords and indentation). Some errors will be tolerated, since you are working on paper. If you forget the name of a Python method, use pseudocode.
> - Unless otherwise stated, you are not required to write code documentation (even though documentation is a good thing!)

Good luck!

1 Python (7 points)

1.1 Questions

a. **Write the code for a function that displays a countdown starting from a value `n`.**

Expected output: `5 4 3 2 1 0 boom.`

```
def compte_a_rebours(n) -> None:
    while n >= 0:
        print(n, end=" ")
        n -= 1
    print("boom.")
```

b. **What will the following code display?**

```
for i in range(10):
    if i == 5:
        break
    if i == 3:
        continue
    print(i)
```

0
1
2
4

c. **Complete the `Velo` class below by creating the following methods:**

- `accelerer`: Method to increase the bicycle's speed by a value `v`.
- `ralentir`: Method to decrease the bicycle's speed by a value `v`.
- `installer_porte_bagage`: Method to install a luggage rack on the bicycle.

```
class Velo:
    def __init__(self, couleur, porte_bagage=False):
        self.couleur = couleur
        self.vitesse = 0
        self.porte_bagage = porte_bagage

    def accelerer(self, v: int) -> None:
        self.vitesse += v

    def ralentir(self, v: int) -> None:
        self.vitesse = max(0, self.vitesse - v)

    def installer_porte_bagage(self) -> None:
        self.porte_bagage = True
```

d. **Create a blue bicycle. Install a luggage rack on it and make it travel at 20 km/h.**

```
v = Velo("Bleu")
v.installer_porte_bagage()
v.accelerer(20)
```

e. **Write the code for a function that takes a list as a parameter and returns that list without duplicates.**

```
def sans_doublons(liste):
    res = []
    for v in liste:
        if v not in res:
            res.append(v)
    return res
```

or simply by using sets:

```
def sans_doublons(liste):
    return list(set(liste))
```

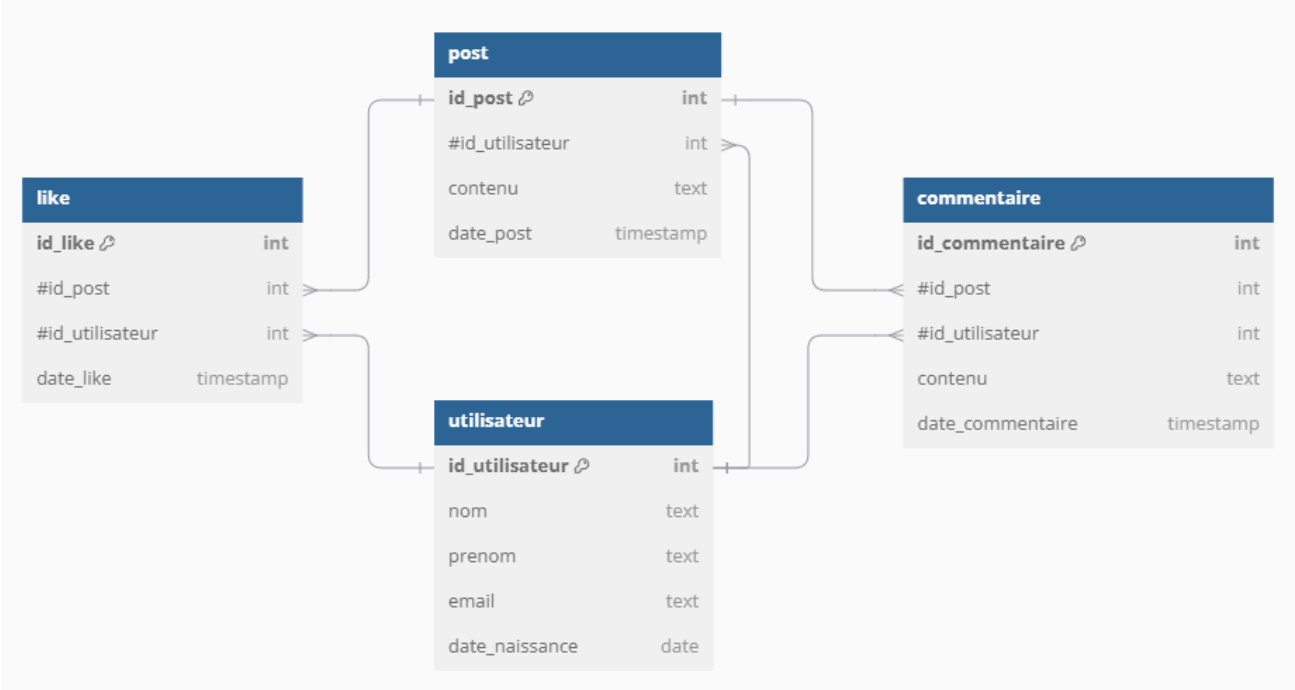
f. **Write the code for a function that takes a string as input and returns the number of occurrences of each letter.**

```
def compter_lettres(chaine: str) -> dict:
    occurrences = {}
    for c in chaine:
        if c.isalpha():
            if c in occurrences:
                occurrences[c] += 1
            else:
                occurrences[c] = 1
    return occurrences
```

2 SQL (7 points)

We model the data of a very simple social network where users can:

- create posts
- like posts
- comment on posts



Write the SQL query that answers each question below.

a. **List all users (last name and first name) in alphabetical order.**

```sql
SELECT nom, prenom
  FROM utilisateur
 ORDER BY nom, prenom;
```

b. **List the contents of comments containing both the letter Z and the letter W.**

```sql
SELECT contenu
  FROM commentaire
 WHERE contenu LIKE '%Z%'
   AND contenu LIKE '%W%';
```

c. **Which posts (content and username) are there where the user commented on a post they themselves wrote?**

```sql
SELECT p.contenu,
       u.nom,
       u.prenom
  FROM post p
  JOIN commentaire c USING(id_post)
```

```
  JOIN utilisateur ON p.id_utilisateur = u.id_utilisateur
 WHERE c.id_utilisateur = p.id_utilisateur;
```

d. **How many comments did each user write? Include users who have never written a comment.**

```
SELECT u.nom,
       u.prenom,
       COUNT(1) AS nb_commentaires
  FROM utilisateur u
  LEFT JOIN commentaire c USING(id_utilisateur)
 GROUP BY u.nom, u.prenom
```

e. **How can you check that no user was able to like the same post twice?**

```
SELECT id_utilisateur,
       id_post,
       COUNT(*) AS nb_likes
  FROM like
 GROUP BY id_utilisateur, id_post
HAVING COUNT(*) > 1;
```

f. **How can you check that a user was not able to like a post they wrote themselves?**

```
SELECT l.id_like,
       l.id_utilisateur,
       l.id_post
  FROM like l
  JOIN post p USING(id_post)
 WHERE l.id_utilisateur = p.id_utilisateur;
```

g. **Delete the post with id 8888.**

```
DELETE FROM post
 WHERE id_post = 8888;
```

Some SQL keywords: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `ASC`, `DESC`, `JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`, `ON`, `USING`, `INSERT INTO`, `UPDATE`, `SET`, `DELETE FROM`, `CREATE TABLE`, `ALTER TABLE`, `ADD COLUMN`, `DROP TABLE`, `DISTINCT`, `AS`, `COUNT`, `AVG`, `SUM`, `MAX`, `MIN`, `AND`, `OR`, `BETWEEN`, `LIKE`, `IS NULL`

3 UML and OOP (6 points)

A medical practice wants to develop an application to manage its doctors, patients, and appointments.

A Doctor has an identifier, a last name, a first name, and a graduation year.

Among the doctors, there are two categories:

- General Practitioner, with an additional attribute indicating whether they make home visits.
- Specialist, who has a special consultation fee.

A Patient is defined by a last name, a first name, a NIR (French social security number), and optionally a primary care physician. A patient must be able to declare, modify, or remove their primary care physician.

An appointment is characterized by a date, a time, a doctor, a patient, and a status (upcoming, completed, cancelled, no-show).

a. **Propose a UML class diagram representing these entities and their relationships.**

b. **Write the constructor of the `Generaliste` class.**

```
class Generaliste(Medecin):
              def    __init__(self,    id_medecin,    nom,    prenom,    annee_diplome,
deplacement_domicile=False):
        super().__init__(id_medecin, nom, prenom, annee_diplome)
        self.deplacement_domicile = deplacement_domicile
```

c. **Write the code for the method that allows a patient to declare a primary care physician.**

```
class Patient:
    def declarer_medecin_traitant(self, medecin):
        self.medecin_traitant = medecin
```