Correction du Bac blanc NSI de janvier 2024

M. Tonnelier

February 11, 2024

Nota Bene : Le total des points des 3 exercices est de 40 points. La note sur 20 est donc la note totale divisée par 2.

Exercice I : Programmation objet en langage Python → 6 points

Question I.1. Ajouter du beurre → 2 points

```
def ajouter_beurre(self, qt) :
  self.qt_beurre += qt
```

Question I.2. Afficher le stock → 2 points

```
def afficher(self) :
  print(f"farine : {self.qt_farine}")
  print(f"oeuf: {self.nb_oeufs}")
  print(f"beurre: {self.qt_beurre}")
```

Question I.3. Vérifier si on peut faire une brioche → 2 points

```
def stock_suffisant_brioche(self) :
  return self.qt_farine >= 350 and self.qt_beurre >= 175 and self.nb_oeufs >= 4
```

Question I.4. Produire une brioche

I.4.a. Nombre de brioches produites → 2 points

Cette instruction n'affiche aucune valeur dans la console car il n'y a pas d'appel à la fonction *print*, mais retourne une valeur : le nombre de brioches produites, à savoir : 2.

I.4.b. Afficher le stock après production → 1,5 point

Cette instruction va afficher :

```
farine: 300
oeuf: 2
beurre: 650
```

Explications supplémentaires : En effet on peut calculer qu'avec 10 œufs au départ, on peut fabriquer que 2 brioches. Après la fabrication de ces 2 brioches, le nombre d'œufs aura été diminué 2 fois de 2, soit 8 œufs en moins. Il n'en reste plus que 2, ce qui est insuffisant pour la condition de continuité de la boucle `while`.
On en déduit que le le stock initial de 1000 kg de farine passe à 300 g :

$$1000 - 2 \times 350 = 300 \tag{1}$$

Et que le stock initial de 1000 g de beurre passe à 650 g :

$$1000 - 2 \times 175 = 650 \tag{2}$$

, après la fabrication de ces 2 brioches.

Question I.5. Nombre total de brioches produites sur tous les lieux de production → 2,5 point

```
def nb_brioches(liste_stocks) :
  total = 0
  for stock in liste_stocks :
    total += stock.produire()
  return total

```

Exercice II : Bases de données → 8 points

Question II.1. Contraintes d'intégrité

II.1.a. Clé primaire → 1 point

Je propose comme clé primaire pour la relation `Mesures`, un attribut unique permettant d'identifier un et un seul tuple de la table. En observant le contenu de la relation `Mesures` on constate que l'attribut `id_mesure` n'est pas répété. Et c'est bien un identifiant d'une seule mesure qu'il nous fallait.

II.1.b. Clé étrangère → 1 point

On peut lier l'attribut `id_centre` de la table `Mesures` avec l'attribut `id_centre` de la table `Centres`.

II.1.c. Contraintes de domaine → 0,5 point

Ce que l'on appelle "domaine" dans le vocabulaire des bases de données, ce sont les types des valeurs que peuvent prendre les attributs.
Voici des exemples de contraintes d'intégrité de domaine de la relation Centres que l'on peut citer :

- `id_centre` de la relation `Centres` est un entier (type `INT`), tout comme `id_centre` de la relation `Centres` qui y fait référence.
- Les coordonnées de position des centres sont des nombres réels de type `FLOAT`. Tout comme l'attribut `pluviometrie` de la relation `Mesures`.

Question II.2. Requêtes sur les centres

II.2.a. Affichage → 1 point

La requête va afficher tous les centres dépassant 500 m d'altitude, à savoir, en l'occurrence :

id_centre	nom_ville	latitude	longitude	altitude
138	Grenoble	45.185	5.723	550
185	Tignes	45.469	6.909	2594
126	Le Puy-en-Velay	45.042	3.888	744
317	Gérardmer	48.073	6.879	855

II.2.b. Filtre par altitude → 3 points

```
SELECT nom_ville
FROM Centres
WHERE altitude > 700 AND altitude < 1200 ;
```

II.2.c. Filtre par longitude et tri par nom → 3 points

```sql
SELECT longitude, nom_ville
FROM Centres
WHERE longitude > 5
ORDER BY nom_ville ASC ;
```

Question II.3. Requêtes sur les mesures

II.3.a. Filtre par date → 1,5 point

```sql
SELECT *
FROM Mesures
WHERE date="2021-10-30"
```

II.3.b. Ajout d'un tuple → 1 point

```sql
INSERT INTO TABLE Mesures
VALUES(3650, 138, "2021-11-08", 11.0, 1013, 0) ;
```

Question II.4. Requêtes complexes

II.4.a. Double filtre par latitude minimale → 1 point

Cette requête SQL va afficher tous les attributs du (ou des) centre(s) ayant la latitude la plus basse, en l'occurrence : La requête va afficher tous les centres dépassant 500 m d'altitude, à savoir, en l'occurrence :

id_centre	nom_ville	latitude	longitude	altitude
456	Nice	43.706	7.262	260

II.4.b. Jointure → 2 points

```sql
SELECT DISTINCT nom_ville
FROM Mesures
JOIN Centres ON Mesures.id_centre = Centres.id_centre
WHERE date LIKE "2021-10-%"
and temperature < 10 ;
```

Exercice III : Structures de données (dictionnaires) → 6 points

Question III.1. Chiffrer un message

III.1.a. Retrouver une valeur à partir d'une clé → 1 point

À la clé `"D"` est associée la valeur `"C"`. En python on y accède avec la syntaxe :

```
alpha["D"]
```

III.1.b. Déchiffrer manuellement → 0,5 point

Une fois chiffrée, la chaîne de caractères `"BAGAGE"` devient `"DBEBEF"`.

Question III.2. Chiffrer un mot automatiquement → 4 points

```
def chiffrer(mot, alpha):
  res = ""
  for lettre in mot:
    res += alpha[lettre]
  return res
```

Question III.3. Déchiffrer un mot

III.3.a. Dictionnaire inversé → 1 point

Il s'agit du dictionnaire `dico` mais où les clés et les valeurs sont inversées, à savoir :

```
dico_inverse = {"B":"A", "D":"B", "A":"C", "C":"E", "F":"E", "G":"F", "E":"G"}
```

III.3.b. Inverser le dictionnaire automatiquement → 1,5 point

```
def dico_dechiffrement(dico):
  nouveau = {}
  for lettre in dico:
    code = dico[lettre]
    nouveau[code] = lettre
  return nouveau
```

III.3.c. Déchiffrer un mot automatiquement → 4 points

```
def dechiffre(mot, dico):
  dico2 = dico_dechiffrement(dico)
  return chiffrer(mot, dico2)
```