NOM, Prénom :

Contrôle 02

NSI2

Exercice 1

bases de données, langage SQL

L'énoncé de cet exercice utilise les mots du langage SQL suivants :

```
SELECT FROM, WHERE, JOIN ON, INSERT INTO VALUES
UPDATE, SET, DELETE, COUNT, AND, OR
```

Pour la gestion des réservations clients, on dispose d'une base de données nommée `gare` dont le schéma relationnel est le suivant :

`Train (numT, provenance, destination, horaireArrivee, horaireDepart)`

`Reservation (numR, nomClient, prenomClient, prix, numT)`

Les attributs soulignés en trait plein sont des clés primaires. L'attribut souligné en pointillés est une clé étrangère : `Reservation.numT` fait référence à la clé primaire `Train.numT`.

Les attributs `horaireDepart` et `horaireArrivee` sont de type `TIME` et s'écrivent selon le format `hh:mm`, où `hh` représente les heures et `mm` les minutes.

1. Quel nom générique donne-t-on aux logiciels qui assurent, entre autres, la persistance des données, l'efficacité de traitement des requêtes et la sécurisation des accès pour les bases de données ?

2. a. On considère les requêtes SQL suivantes :

SQL

```sql
DELETE FROM Train WHERE numT = 1241 ;
DELETE FROM Reservation WHERE numT = 1241 ;
```

Sachant que le train n°1241 a été enregistré dans la table `Train` et que des réservations pour ce train ont été enregistrées dans la table `Reservation`, expliquer pourquoi

cette suite d'instructions renvoie une erreur.

b. Citer un cas pour lequel l'insertion d'un enregistrement dans la table `Reservation` n'est pas possible.

3. Écrire des requêtes SQL correspondant à chacune des instructions suivantes :

a. Donner tous les numéros des trains dont la destination est « Lyon ».

b. Ajouter une réservation n° 1307 de 33 € pour M. Alan Turing dans le train n° 654.

c. Suite à un changement, l'horaire d'arrivée du train n° 7869 est programmé à 08 : 11. Mettre à jour la base de données en conséquence.

d. Produire la table des noms et prénoms de tous les clients qui ont effectué une réservation pour un train partant de Paris et allant à Marseille.

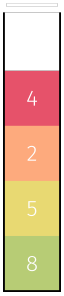
Exercice 2

piles

Interface de la pile

- `cree_pile_vide()` créée une pile vide ;
- `empiler(pile, valeur)` empile la valeur sur la pile ;
- `depiler(pile)` renvoie la valeur sur la pile et l'enlève de la pile ;
- `est_vide()` indique si la pile est vide on non ;

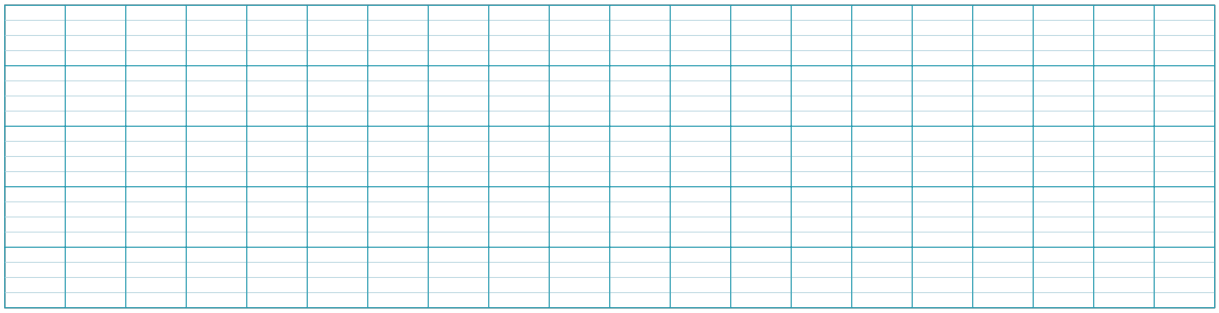
1. On suppose dans cette question que le contenu d'une pile `p` est le suivant (les éléments étant empilés par le haut) :



Quel sera le contenu de la pile `q` après exécution de la suite d'instructions suivante ?

Python

```
q = creer_pile_vide()
while not est_vide(p):
      empiler(q, depiler(p))
```



On appelle *hauteur* d'une pile le nombre d'éléments qu'elle contient. La fonction `hauteur_pile` prend en paramètre une pile `p` et renvoie sa hauteur.
Après appel de cette fonction, la pile `p` doit avoir retrouvé son état d'origine.
Exemple : si `p` est la pile de la question 1, `hauteur_pile(p)` vaut 4.

2. Compléter le programme Python suivant implémentant la fonction `hauteur_pile`.

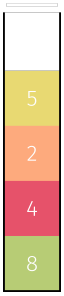
Python

```
def hauteur_pile(p):
      q = creer_pile_vide()
      n = 0
      while not est_vide(p):
            ...
            x = depiler(p)
            empiler (q, x)
      while not est_vide(q):
            ...
            empiler(p, x)
      return ...
```

3. Créer une fonction `max_pile` ayant pour paramètres une pile `p` et un entier `i`.
Cette fonction renvoie la position `j` de l'élément maximum parmi les `i` derniers éléments empilés de la pile `p` . Après appel de cette fonction, la pile `p` devra avoir retrouvé son état d'origine.
La position du sommet de la pile est 1 (et non 0).
Exemple : si `p` est la pile de la question 1, `max_pile(p, 2)` vaut 1 car c'est l'élément en position 1 (le sommet) de `p` qui est maximal parmi les 2 premiers.

4. Créer une fonction `retourner` ayant pour paramètres une pile `p` et un entier `j`. Cette fonction inverse l'ordre des `j` derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

Exemple : si `p` est la pile de la question 1, après l'appel de `retourner(pile, 3)` l'état de `p` sera :

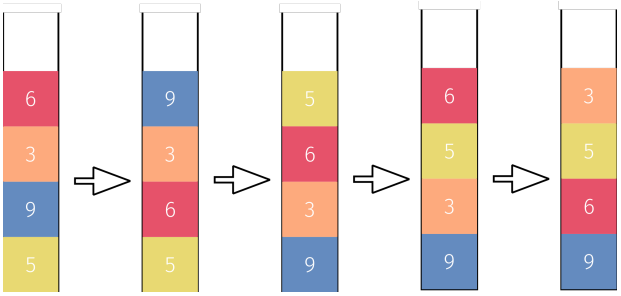


5. L'objectif de cette question est de trier une pile de crêpes.

On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).
On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus. Le principe est le suivant :

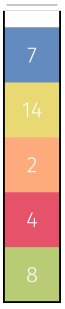
- On recherche la plus grande crêpe.
- On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.
- On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.
- La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

Exemple :

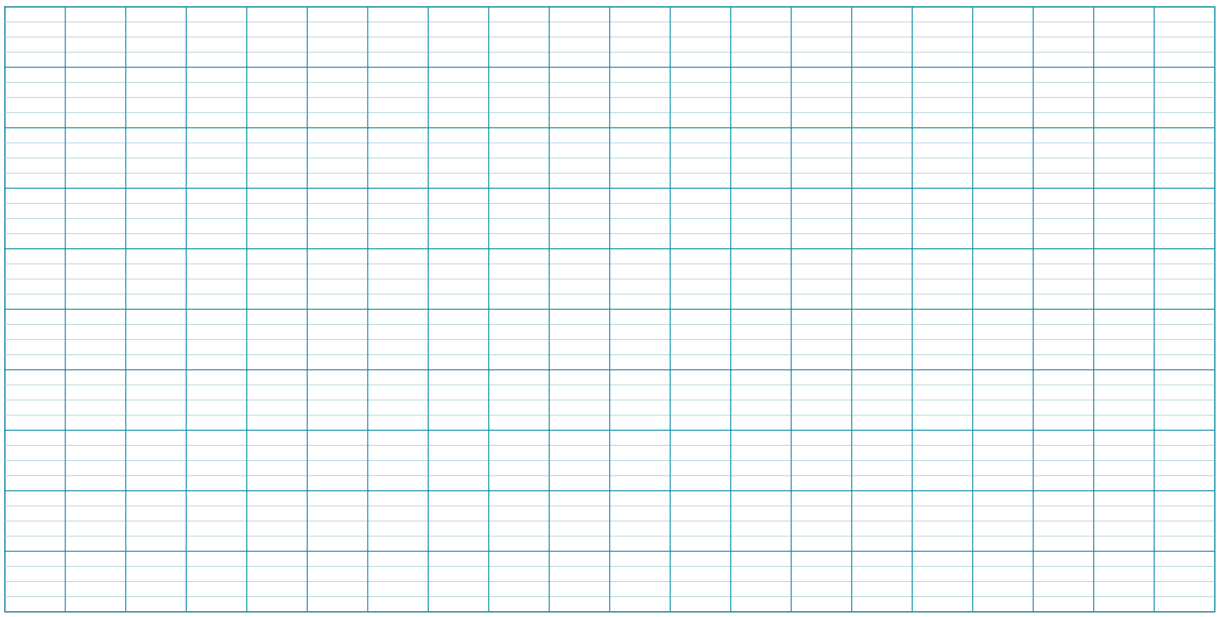


Créer la fonction `tri_crepes` ayant pour paramètre une pile `p` . Cette fonction trie la pile `p` selon la méthode du tri crêpes et ne renvoie rien. On utilisera les fonctions créées dans les questions précédentes.

Exemple : Si la pile `p` est dans l'état



alors `tri_crepes(p)` la met dans l'état



Exercice 1

POO

Les participants à un jeu de LaserGame sont répartis en équipes et s'affrontent dans ce jeu de tir, revêtus d'une veste à capteurs et munis d'une arme factice émettant des infrarouges. Les ordinateurs embarqués dans ces vestes utilisent la programmation orientée objet pour modéliser les joueurs. La classe Joueur est définie comme suit :

Python

```
class Joueur:

      def __init__(self, pseudo, identifiant, equipe):
            self.pseudo = pseudo
            self.equipe = equipe
            self.id = identifiant
            self.nb_de_tirs_emis = 0
            self.liste_id_tirs_recus = []
            self.est_actif = True

      def tire(self):
            '''déclenchée par l'appui sur la gachette'''
            if self.est_actif == True:
                  self.nb_de_tirs_emis = self.nb_de_tirs_emis +
                   ↪ 1

      def est_determine(self):
            '''renvoie True si le joueur réalise un
```

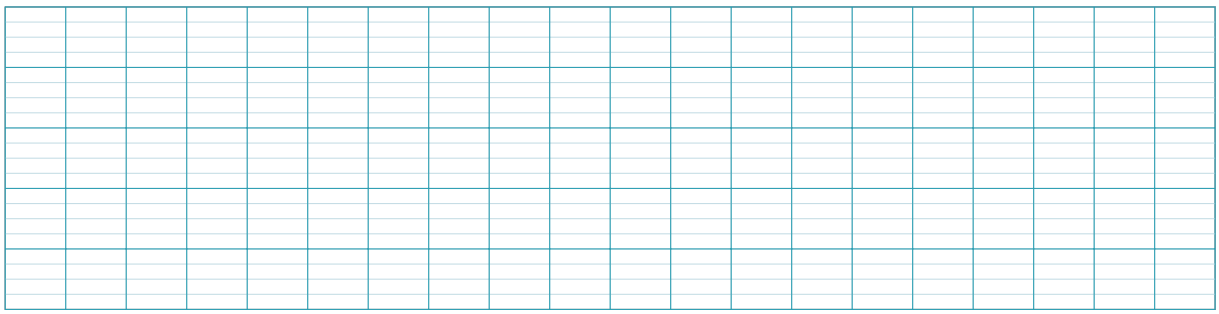
```
            grand nombre de tirs'''
            return self.nb_de_tirs_emis > 500

      def subit_un_tir(self, id_recu):
            '''déclenchée par les capteurs de la veste'''
            if self.est_actif == True:
                  self.est_actif = False
                  self.liste_id_tirs_recus.append(id_recu)
```

1. Parmi les instructions suivantes, entourer celle qui permet de déclarer un objet `joueur1`, instance de la classe `Joueur`, correspondant à un joueur dont le pseudo est `Sniper`, dont l'identifiant est `319` et qui est intégré à l'équipe `A` :

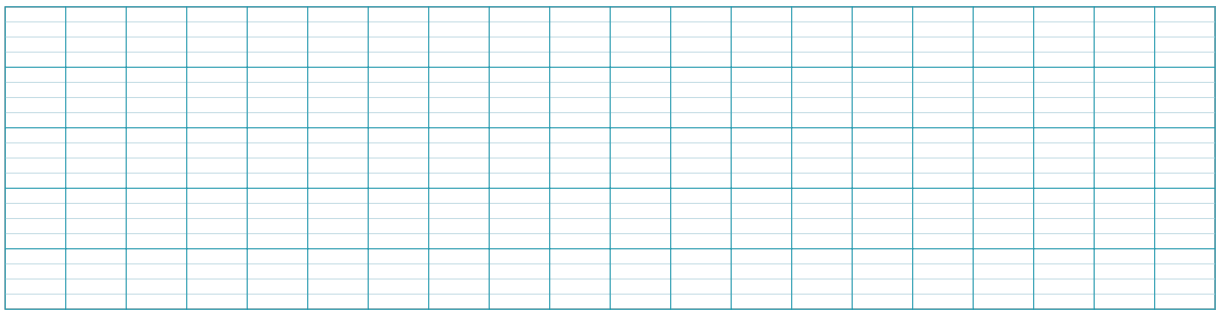
- Instruction 1 :
  `joueur1 = ["Sniper", 319, "A"]`
- Instruction 2 :
  `joueur1 = new Joueur["Sniper", 319, "A"]`
- Instruction 3 :
  `joueur1 = Joueur("Sniper", 319, "A")`
- Instruction 4 :
  `joueur1 = Joueur{"pseudo":"Sniper", "id":319, "equipe":"A"}`

2. La méthode `subit_un_tir` réalise les actions suivantes : Lorsqu'un joueur actif subit un tir capté par sa veste, l'identifiant du tireur est ajouté à l'attribut `liste_id_tirs_recus` et l'attribut `est_actif` prend la valeur `False` (le joueur est désactivé). Il doit alors revenir à son camp de base pour être de nouveau actif.

a. Écrire la méthode `redevenir_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.



b. Écrire la méthode `nb_de_tirs_recus` qui renvoie le nombre de tirs reçus par un joueur en utilisant son attribut `liste_id_tirs_recus`.



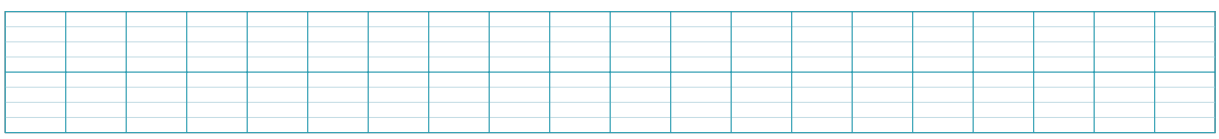
3. Lorsque la partie est terminée, les participants rejoignent leur camp de base respectif où un ordinateur, qui utilise la classe `Base`, récupère les données.
La classe `Base` est définie par :

- ses attributs :
  - `equipe` : nom de l'équipe (str), par exemple, `A` ;
  - `liste_des_id_de_l_equipe` qui correspond à la liste (list) des identifiants connus des joueurs de l'équipe ;
  - `score` : score (int) de l'équipe, dont la valeur initiale est 1000.
- ses méthodes :
  - `est_un_id_allie` qui renvoie `True` si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, `False` sinon ;
  - `incremente_score` qui fait varier l'attribut score du nombre passé en paramètre ;
  - `collecte_information` qui récupère les statistiques d'un participant passé en paramètre (instance de la classe `Joueur`) pour calculer le score de l'équipe.

Python

```
def collecte_information(self,participant):
      if participant.equipe == self.equipe : # test 1
            for id in participant.liste_id_tirs_recus:
                  if self.est_un_id_allie(id): # test 2
                        self.incremente_score(-20)
                  else:
                        self.incremente_score(-10)
```

a. Indiquer le numéro du test (test 1 ou test 2) qui permet de vérifier qu'en fin de partie un participant égaré n'a pas rejoint par erreur la base adverse.



b. Décrire comment varie quantitativement le score de la base lorsqu'un joueur de cette équipe a été touché par le tir d'un coéquipier.

On souhaite accorder à la base un bonus de 40 points pour chaque joueur particulièrement déterminé (qui réalise un grand nombre de tirs).

4. Compléter, en utilisant les méthodes des classes `Joueur` et `Base`, les 2 lignes de codes suivantes qu'il faut ajouter à la fin de la méthode `collecte_information` :

Python

```
#si le participant réalise un grand nombre de tirs
........

#le score de la Base augmente de 40
.........
```