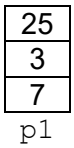
**Spécialité NSI Terminale**
**Correction du**
**devoir commun**
Vendredi 3 février 2023

## Exercice 1

**1.** On obtient le schéma suivant :

**2. a.** La variable tem prend la valeur 25.

**b.** La pile aura le contenu suivant




Elle a donc retrouvé sa valeur initiale après exécution de la fonction.

**3.** On peut écrire le code suivant :

```
def addition(p) :
    x1 = depiler(p)
    x2 = depiler(p)
    empiler(p, x1+x2)
```

**4.** On peut écrire le code suivant :

```
p = pile_vide()
empiler(p, 3)
empiler(p, 5)
addition(p)
empiler(p, 7)
multiplication(p)
```

## Exercice 2

**1. a)**

* Le couple `(nomSport , nomStation)` est la clef primaire de la relation `Sport`.
* `nomStation` est une clef étrangère de la relation `Sport`, qui référence `nomStation` de la relation `Station`.

**b)**

- Contrainte d'intégrité de domaine : l'attribut `prix` de `Sport` doit être un entier.
- Contrainte d'intégrité de relation : les couples `(nomSport , nomStation)` doivent être différents dans chaque enregistrement de la relation `Sport` puisqu'il s'agit de la clef primaire.
- Contrainte d'intégrité de référence : l'attribut `nomStation` de chaque enregistrement de la relation `Sport` doit précédemment exister dans la relation `Station`.

**2. a)** Un enregistrement de la relation `Sport` de clef primaire `("planche à voile", "La tramontane catalane")` existe déjà selon l'énoncé. La requête INSERT INTO a été refusée car elle tente d'ajouter un nouvel enregistrement ayant la même clef primaire.
Pour modifier le tarif de ce séjour, on utilise la requête SQL suivante :

```
UPDATE Sport SET prix=1350
WHERE nomSport="planche à voile" AND nomStation="La tramontane catalane" ;
```

**b)** On peut écrire les requêtes SQL suivantes :

```
INSERT INTO Station VALUES ("Soleil Rouge", "Bastia", "Corse") ;
INSERT INTO Sport VALUES ("plongee", "Soleil Rouge", 900) ;
```

**3. a)** On peut écrire la requête suivante :

```
SELECT mail FROM Client ;
```

**b)** On peut écrire la requête suivante :

```
SELECT nomStation FROM Sport WHERE nomSport="plongee" ;
```

**4. a)** On peut écrire la requête suivante :

```
SELECT ville, nomStation FROM Station
JOIN Sport ON Station.nomStation=Sport.nomStation
WHERE Sport.nomSport="plongee" ;
```

**b)** On peut écrire la requête suivante :

```
SELECT COUNT(*) FROM Sejour
JOIN Station ON Sejour.nomStation=Station.nomStation
WHERE Station.region="Corse" and Sejour.annee=2020 ;
```

## Exercice 3

**1. a.** On doit compléter le code de la façon suivante :

```
def donnePremierIndiceLibre(Mousse) :
    i=0
    while i < 6 and Mousse[i] != None :
          i = i+1
    return i
```

**b.** On peut écrire le code suivant :

```
def placeBulle(B) :
    ind = donnePremierIndiceLibre(Mousse)
    if ind < 6 :
        Mousse[ind] = B
```

**2.** On peut écrire le code suivant :

```
def bullesEnContact(B1, B2) :
  return distanceEntreBulles(B1, B2) <= B1.rayon \
         + B2.rayon
```

ou bien, en décomposant les opérations :

```
def bullesEnContact(B1, B2) :
    dist = distanceEntreBulles(B1, B2)
    somme_rayons = B1.rayon + B2.rayon
    return dist <= somme_rayons
```

**3.** On doit compléter le code de la façon suivante :

```
def collision(indPetite, indGrosse, Mousse) :
    # calcul et mise à jour du nouveau rayon de la grosse
bulle
    surfPetite = pi*Mousse[indPetite].rayon**2
    surfGrosse = pi*Mousse[indGrosse].rayon**2
    nouvSurfaceGrosse = surfPetite + surfGrosse
    nouvRayonGrosse = sqrt(nouvSurfaceGrosse/pi)
    Mousse[indGrosse].rayon = nouvRayonGrosse
    #réduction de 50% de la vitesse de la grosse bulle
    Mousse[indGrosse].dirx = Mousse[indGrosse].dirx/2
    Mousse [indGrosse].diry = Mousse[indGrosse].diry/2
    #suppression de la petite bulle dans Mousse
    Mousse[indPetite] = None
```