TERMINALE GENERALE

ENSEIGNEMENT DE SPECIALITÉ

NUMERIQUE ET SCIENCES INFORMATIQUES

Épreuve du 3 février 2023

Durée : **3 heures 30**

Vous n'êtes pas autorisé à quitter la salle de composition avant une durée de 2 heures après le début de l'épreuve.

L'usage de la calculatrice n'est pas autorisé.

Vous devez traiter les trois exercices du sujet.

Chaque exercice est noté sur 7 points.

Vous indiquerez clairement sur vos copies les numéros des exercices traités.
Vous rédigerez chaque exercice sur une copie différente.

Les éléments éventuellement écrits sur le sujet ne seront pas corrigés : vous devez répondre aux questions uniquement sur vos copies.

La qualité de la rédaction et la précision des réponses sont des éléments importants dans l'appréciation des résultats.

EXERCICE 1 (7 points)

Cet exercice porte sur les structures de données (pile).

La notation polonaise inverse (NPI) permet d'écrire des expressions de calculs numériques sans utiliser de parenthèse. Cette manière de présenter les calculs a été utilisée dans des calculatrices de bureau dès la fin des années 1960. La NPI est une forme d'écriture d'expressions algébriques qui se distingue par la position relative que prennent les nombres et leurs opérations.

Par exemple :

Notation classique	Notation NPI
3+9	3 9 +
8×(3+5)	8 3 5 + ×
(17+5)×4	17 5 + 4 ×

L'expression est lue et évaluée de la gauche vers la droite en mettant à jour une pile.

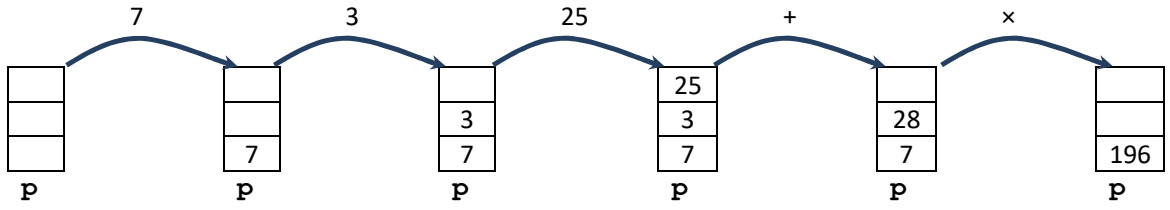
- Les nombres sont empilés dans l'ordre de la lecture.
- Dès la lecture d'un opérateur (+, -, ×, /), les deux nombres au sommet de la pile sont dépilés et remplacés par le résultat de l'opération effectuée avec ces deux nombres. Ce résultat est ensuite empilé au sommet de la pile.

A la fin de la lecture, la valeur au sommet est renvoyée.

Exemple : l'expression `7 3 25 + ×` qui correspond au calcul $7 \times (3 + 25)$ s'évalue à 196 comme le montrent les états successifs de la pile créée, nommée **p** :

- On empile la valeur 7.
- On empile la valeur 3.
- On empile la valeur 25.
- On remplace les deux nombres du sommet de la pile (25 et 3) par leur somme 28.
- On remplace les deux nombres du sommet de la pile (28 et 7) par leur produit 196.

Schéma descriptif des différentes étapes d'exécution.



1. En vous inspirant de l'exemple ci-dessus, dessiner le schéma descriptif de ce que donne l'évaluation par la NPI de l'expression `12 4 5 × +`.

2. On dispose de la pile suivante nommée `p1` :

On rappelle ci-dessous les primitives de la structure de pile (LIFO : Last In First out) :

Fonction	Description
pile_vide()	Créé et renvoie une nouvelle pile vide
empiler(p, e)	Place l'élément **e** au sommet de la pile **p**.
depiler(p)	Supprime et renvoie l'élément se trouvant au sommet de **p**.
est_vide(p)	Renvoie un booléen indiquant si **p** est vide ou non.

On dispose aussi de la fonction suivante, qui prend en paramètre une pile `p` :

```
def top(p) :
    x = depiler(p)
    empiler(p, x)
    return x
```

On exécute la ligne suivante `temp = top(p1)` :

a. Quelle valeur contient la variable temp après cette exécution ?

b. Représenter la pile `p1` après cette exécution.

3. En utilisant uniquement les 4 primitives d'une pile, écrire en langage Python la fonction `addition(p)` qui prend en paramètre une pile `p` d'au moins deux éléments et qui remplace les deux nombres du sommet d'une pile `p` par leur somme.

Remarque : cette fonction ne renvoie rien, mais la pile `p` est modifiée.

4. On considère que l'on dispose également d'une fonction `multiplication(p)` qui remplace les deux nombres du sommet d'une pile `p` par leur produit (on ne demande pas d'écrire cette fonction). Recopier et compléter, en n'utilisant que les primitives d'une pile et les deux fonctions `addition` et `multiplication`, la suite d'instructions (ci-dessous) qui réalise le calcul $(3 + 5) \times 7$ dont l'écriture en NPI est : **3 5 + 7 ×**.

```
p = pile_vide()
empiler(p, 3)
...
```

EXERCICE 2 (7 points)

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

L'énoncé de cet exercice utilise les mots du langage SQL suivants :
`SELECT, FROM, WHERE, JOIN, INSERT INTO, VALUES, UPDATE, SET, COUNT.`
On rappelle qu'en SQL, la fonction d'agrégation `COUNT` permet de compter le nombre d'enregistrements dans une table.

L'entreprise "Vacances autrement" propose des séjours d'une durée d'une semaine dans différentes stations situées en France. Les séjours sont organisés autour d'une seule activité sportive. Par exemple, la station "La Tramontane Catalane" située à Leucate propose une formule autour de la planche à voile. Les clients pourront durant toute la durée du séjour pratiquer ce sport : le matériel est fourni et deux sessions encadrées par des moniteurs diplômés sont organisées chaque journée. Cette même station propose aussi des séjours pour les amateurs de kitesurf.
Les réservations se font par l'intermédiaire d'un site internet : chaque client complète un formulaire avec ses données personnelles (nom, prénom...) et choisit le séjour souhaité.
Ce site s'appuie sur une base de données relationnelle. Elle contient notamment les relations `Station, Sport, Client` et `Sejour`.

Voici le schéma relationnel où les clefs primaires sont soulignées et les clefs étrangères sont suivies du symbole `#` :

```
Station (nomStation, ville, region)
Sport (nomSport, nomStation#, prix)
Client (cID, nom, prenom, mail)
Sejour (cID, semaine, annee, nomStation#,nomSport#)
```

- Les différentes stations sont stockées dans la relation `Station`. Chaque station est identifiée par un nom caractéristique (`nomStation`). Les attributs `ville` et `region` permettent de décrire la localisation de chacune des stations.
- La relation `Sport` fait le lien entre les stations et les activités sportives qui y sont proposées. Les trois attributs sont `nomSport, nomStation` et `prix`. L'attribut `prix` est un nombre entier : il correspond au montant plein tarif (en euros) du séjour d'une semaine dans la station avec le sport associé.
- La relation `Client` caractérise les clients de l'entreprise : ils sont identifiés par un nombre entier (`cID`) et décrits par leur `nom`, `prénom` et adresse `mail`.
- La relation `Sejour` permet de lister les séjours auxquels les clients ont participé. Les deux attributs `semaine` et `annee` sont des nombres entiers : ils permettent d'identifier le moment où a été effectué chaque séjour (`semaine` est le numéro de la semaine pendant laquelle le séjour a été réalisé).
  Exemple : un séjour ayant été effectué la semaine numéro 12 de l'année 2020 correspond à `semaine = 12` et à `annee = 2020`.

On donne aussi un extrait des trois premières lignes de chacune de ces relations :

`Station` :

nomStation	ville	region
La tramontane catalane	Leucate	Occitanie
La baie sauvage	La Torche	Bretagne
La pinède	Calvi	Corse

`Sport` :

nomSport	nomStation	prix
planche à voile	La tramontane catalane	1200
kitesurf	La tramontane catalane	1100
plongée	La baie sauvage	950

`Client` :

cID	nom	prenom	mail
1	GENEREUX	Eric	eric.genereux@mail.fr
2	PIERRE	Daniel	daniel.pierre@mail.fr
3	JOLY	Emilie	emilie.joly@mail.fr

`Sejour` :

cID	semaine	annee	nomStation	nomSport
1	26	2020	La tramontane catalane	planche à voile
1	38	2020	La tramontane catalane	planche à voile
2	33	2020	La baie sauvage	plongée

1. a) Donner la clé primaire et les éventuelles clés étrangères de la relation `Sport`.

   b) Citer une contrainte d'intégrité de domaine puis une contrainte d'intégrité de relation et enfin une contrainte d'intégrité de référence que doivent respecter les données de la relation `Sport`.

2. a) Le tarif du séjour à la station "La tramontane catalane" basé sur la planche à voile passe de 1 200 euros à 1 350 euros.
   La requête SQL suivante a été utilisée pour la mise à jour du tarif :

   ```
   INSERT INTO Sport VALUES ("planche à voile",
                             "La tramontane catalane",1350) ;
   ```

   Cette requête a été rejetée et la mise à jour n'a pas été effectuée.
   Après avoir expliqué pourquoi cette requête a été refusée, proposer une requête SQL qui permettra de modifier le tarif de ce séjour.

   b) Une nouvelle station vient d'être référencée. Son nom est "Soleil Rouge". Elle est située à Bastia en Corse. Des séjours d'une semaine y seront organisés. Le sport pratiqué sera la plongée au tarif de 900 euros.
   Écrire les requêtes SQL permettant d'insérer ces nouvelles données dans la base.

3. a) L'agence souhaite envoyer un mail d'information à ses clients pour leur présenter cette nouvelle possibilité de séjour mise en place à Bastia. Écrire une requête SQL permettant d'obtenir l'adresse mail de tous les clients.

   b) Un client qui pratique la plongée souhaite réserver un séjour. Écrire une requête SQL permettant d'obtenir le nom de toutes les stations où l'on peut pratiquer la plongée.

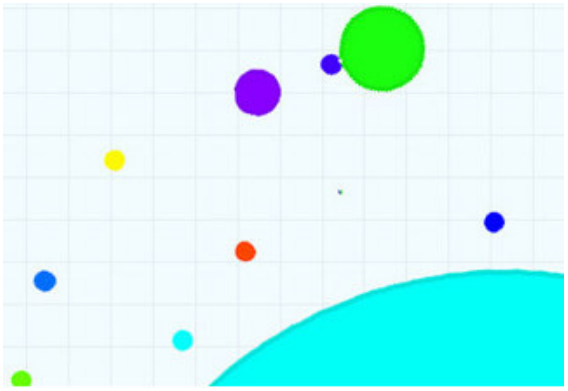
4. a) Pour faire son choix, le client souhaiterait connaître les villes où sont situées les stations dans lesquelles il pourra pratiquer la plongée.
   Écrire une requête SQL permettant d'obtenir le nom des villes ainsi que le nom des stations où l'on peut pratiquer la plongée.

   b) Écrire une requête SQL permettant de déterminer le nombre total de séjours effectués en Corse durant l'année 2020.

EXERCICE 3 (7 points)

Cet exercice porte sur les structures de données (programmation objet).

Dans un jeu de plateforme, des bulles de couleurs et de diamètres différents se déplacent de manière aléatoire. A chaque fois qu'une bulle touche une bulle plus grande, la petite cède son contenu à la plus grande, et donc celle-ci augmente de surface. Par exemple, si une bulle de 1 cm^2 rencontre une bulle de 4 cm^2, la petite bulle disparait et la plus grande a désormais une surface de 5 cm^2. A chaque collision, la vitesse de la grande bulle est réduite de moitié.



Le développeur a choisi de coder en Python, chaque bulle est un objet disposant entre autre des attributs suivants :

* `xc`, `yc` sont deux entiers, les coordonnées du pixel placé au centre de la bulle,
* `rayon` est un entier, le rayon de la bulle en pixels,
* `couleur` est un entier, la couleur de la bulle,
* `dirx`, `diry` sont deux décimaux (`float`) qui déterminent les déplacements à l'horizontale et à la verticale à chaque fois que la bulle se déplace. Ces deux valeurs déterminent donc la direction et la vitesse de la bulle. Par exemple si `dirx` vaut 0.5 et `diry` vaut 0.0, la bulle se déplace vers la droite uniquement alors que si `dirx` vaut -1.0 et `diry` vaut 0.0, la bulle se déplace vers la gauche et deux fois plus vite que précédemment.

On suppose que toutes les fonctions de la bibliothèque `math` ont déjà été importées par l'instruction `from math import *`.
La fonction `randint` de la bibliothèque `random` prend en paramètre deux entiers et renvoie un entier aléatoire dans la plage définie par les deux paramètres.
Exemple : `randint(-1, 5)` peut renvoyer une des valeurs suivantes : $-1, 0, 1, 2, 3, 4, 5$.

1. Pour simplifier, on se limitera à un jeu de six bulles. Au départ, on crée une liste appelée `Mousse` de longueur six contenant six emplacements vides :

```
Mousse = [None, None, None, None, None, None]
```

Le code ci-dessous montre le début du programme et notamment la structure définition de la classe nommée `Cbulle` ainsi que le code permettant le déplacement d'une bulle.

```
from random import randint
from math import *

class Cbulle:
  def __init__(self):
    self.xc = randint(0, 100)
    self.yc = randint(0, 100)
```

```
    self.rayon = randint(0, 10)
    self.dirx = float(randint(-1, 1)) # dirx et diry valent
    self.diry = float(randint(-1, 1)) # -1.0 ou 0.0. ou 1.0
    self.couleur = randint(1,65535)

  def bouge(self):
     # déplace la bulle
     self.xc = self.xc + self.dirx
     self.yc = self.yc + self.diry
```

On crée les six bulles une à une et ces objets sont stockés dans les emplacements vides de la liste `Mousse`.

```
Mousse = [bulle1, bulle2, bulle3, bulle4, bulle5, bulle6]
```

Lors d'une collision, la bulle la plus petite disparait et est remplacée dans la liste par la valeur `None` tandis que la plus grosse a sa surface qui augmente.

Au cours d'une partie, si une ou plusieurs bulles ont disparue, le programme peut en introduire de nouvelles dans le jeu: dans ce cas, lorsqu'une nouvelle bulle apparaît, elle remplace le premier `None` de la liste `Mousse`.

a. Recopier les quatre dernières lignes et compléter les `...` du code python ci-dessous.

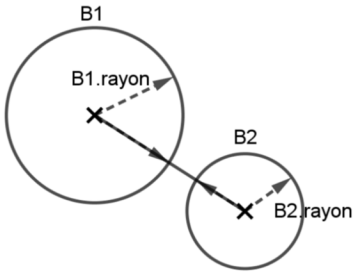
```
def donnePremierIndiceLibre(Mousse):
    """
      Mousse est une liste.
      La fonction doit renvoyer l'indice du premier
      emplacement libre (contenant None) dans la liste
      Mousse ou renvoyer 6 en l'absence d'un emplacement
      libre dans Mousse.
    """
    i=0
    while ... and Mousse[i] != None :
          ...
    return i
```

b. Lorsque le jeu crée une bulle (instance de la classe `Cbulle`), il doit ensuite la placer dans la liste `Mousse` à la place d'un `None`.

Écrire la fonction `placeBulle(B)` qui reçoit en paramètre un objet de type `Cbulle` et qui place cet objet dans la liste `Mousse`. Cette fonction ne renvoie rien, mais la liste `Mousse` est modifiée. Si aucun emplacement n'est disponible, la fonction ne modifie rien.

2. Pour le bon déroulement du jeu, on a besoin aussi d'une fonction `bullesEnContact(B1, B2)` qui renvoie `True` si la bulle B2 touche la bulle `B1` et `False` dans le cas contraire.

On peut remarquer que deux bulles sont en contact si la distance qui sépare leur centre est inférieure ou égale à la somme de leurs rayons.



On dispose de la fonction `distanceEntreBulles(B1, B2)` qui calcule et renvoie la distance entre les centres de bulles B1 et B2.
Écrire la fonction `bullesEnContact(B1, B2)`.

3. Quand une petite bulle touche une plus grosse bulle, on appelle la fonction `collision`, ci-dessous, où `indPetite` est l'indice de la petite bulle et `indGrosse` l'indice de la grosse bulle dans `Mousse`.

Recopier et compléter les `...` de la fonction `collision`.

```
def collision(indPetite, indGrosse, mousse) :
    """
      Absorption de la plus petite bulle d'indice indPetite
      par la plus grosse bulle d'indice indGrosse. Aucun test
      n'est réalisé sur les positions.
    """
    # 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 = ...
    nouvRayonGrosse = sqrt(nouvSurfaceGrosse/pi)
    ... = nouvRayonGrosse
    #réduction de 50% de la vitesse de la grosse bulle
    Mousse[indGrosse].dirx = ...
    Mousse [indGrosse].diry = ...
    #suppression de la petite bulle dans Mousse
    ...
```