| Devoir TERMINALE NSI n°2 | |
|---|---|
| **Pile, File, POO** | |
| Terminale – spécialité NSI | **Octobre 2020** |
| *Durée 2H* | |
| *Calculatrice interdite, mémento Python autorisé* | |

## Epreuve écrite n°1

*durée : environ 3 /4h*

**QCM** (*3 points*))

1. Quelle opération ne fait pas partie de l'interface d'une pile ?
   - ☐ Ajouter un élément à la pile
   - ☐ Retirer l'élément le plus récent de la pile
   - Retirer l'élément le plus ancien de la pile

   Dans une pile, c'est le dernier élément ajouté qui peut-être enlevé, et pas le plus ancien

2. Quelle opération ne fait pas partie de l'interface d'une file ?
   - ☐ Ajouter un élément à la file
   - Retirer l'élément le plus récent de la file
   - ☐ Retirer l'élément le plus ancien de la file

   Dans une file, c'est le premier élément ajouté qui part en premier.

3. L'opération `défiler` d'une file s'exécute en un temps qui est proportionnel au nombre de valeurs stockées dans la file
   - ☐ Vrai
   - Faux : Le temps mis pour retirer l'élément le plus ancien de est constant et indépendant de la taille de la liste

4. On souhaite écrire une portion de code qui permette de savoir su une année est bissextile ou non. Qu'est ce qui est le plus approprié ?
   - ☐ Ecrire un programme principal qui demande à l'utilisateur de taper une année et qui indique si elle est bissextile
   - Ecrire une fonction qui indique si une année passée en paramètre est bissextile ou non en renvoyant un booléen
   - ☐ Ecrire un module bissextile.py qui contiendra tout ce qu'il faut pour déterminer le caractère bissextile d'une année
   - ☐

5. Quel message d'exception s'affiche si on tente d'exécuter le code suivant ?

```
a = 1
for i in range(3) :
    print ("i={}, a={}".format(i,a)
    a = 2 * abs
```

   - ☐ NameError
   - ☐ IndexError
   - SyntaxError (manque une parenthèse)
   - ☐ IndentationError

6. Le code suivant calcule les 10 premiers termes de la suite de Fibonacci. Quel message d'exception s'affiche si on tente de l'exécuter ?

```
f=[0,1,0,0,0,0,0,0,0,0]
for i in range(1,10) :
    f[i+1]=f[i] + f[i-1]
print(f)
```

   - ☐ NameError
   - ☐ SyntaxError
   - IndexError (à cause de f[i+1])
   - ☐ IndentationError

**Q01)** - (*4 points*) Écrire la classe Cercle qui permettra au code ci-dessous de fonctionner correctement.

```
# Définition de la classe à faire...
# ...

ca = Cercle(3)                      # Création d'un cercle de rayon=3
cb = Cercle(7)                      # Création d'un cercle de rayon=7
print(ca.calculer_perimetre())      # Calcul du périmètre du cercle
print(ca.calculer_aire())           # Calcul de l'aire du cercle
ca.agrandir(1)                      # Agrandissement du cercle rayon=rayon+1
print(ca.get_rayon())               # Affichage du rayon du cercle
cb.diminuer(2)                      # Diminution du cercle rayon=rayon-2
print(ca.etre_plus_grand_que(cb))   # Comparaison de l'aire des 2 cercles
```

```
class Cercle:
    """Classe Cercle, proposition de corrigé"""
    def __init__(self, rayon):
        self.rayon = rayon

    def get_rayon(self):
        return self.rayon

    def calculer_perimetre(self):
        return 2 * 3.14 * self.rayon

    def calculer_aire(self):
        return 3.14 * self.rayon * self.rayon

    def agrandir(self, agrandissement):
        self.rayon += agrandissement

    def diminuer(self, reduction):
        if self.rayon - reduction > 0:
        self.rayon -= reduction

    def etre_plus_grand_que(self, autre_cercle):
        if self.calculer_aire() > autre_cercle.calculer_aire():
            return True
        else:
            return False
```

**Q02)** - (*2 points*) A l'aide de la méthode assert, écrire les tests unitaires qui permettraient de valider les méthodes __init()__, agrandir() et get_rayon() de la classe Cercle de la question précédente.

```
# Tests unitaires avec assert
ca = Cercle(3)
cb = Cercle(7)
assert ca.get_rayon() == 3, "rayon => 3"
ca.agrandir(1)
assert ca.get_rayon() == 4, "rayon => 4"
```

**Q03)** - (*1 point*) Nommer sur ce document les deux structures de données schématisées dans le tableau ci-dessous :

figure 1 : FILE



figure 2 : PILE

**Q04)** - (*6 points*) On souhaite simuler un train et ses wagons en utilisant une structure de données de type pile. Compléter le code ci-dessous :

```
class Train:
"""Classe pour simuler un train"""
     def __init__(self, nom):
     """Constructeur"""
          self.queue_du_train = None
          self.compteur_wagons = 0

     def etre_vide(self):
     """Pour savoir si le train est vide ou pas"""
          if self.queue_du_train is None:
               return True
          else:
               return False

     def empiler(self, numero, contenu):
     """Ajoute un wagon au train"""
          self.compteur_wagons += 1
          nouveau_wagon = Wagon(numero, contenu)
          nouveau_wagon.set_suivant(self.queue_du_train)
          self.queue_du_train = nouveau_wagon

     def depiler(self):
     """Retire un wagon du train"""
          if self.etre_vide():
               return None
          else:
               self.compteur_wagons -= 1
               wagon = self.queue_du_train.get_wagon()
               self.queue_du_train = self.queue_du_train.get_suivant()
               return wagon

     def calculer_taille(self):
     """Calcule et retourne la taille du train"""
          return self.compteur_wagons

     def acceder_queue(self):
     """Accède au dernier wagon sans le supprimer"""
          if self.etre_vide():
               return None
          else:
               return self.queue_du_train.get_wagon()
```