المعهد التحضيري للدراسات الهندسية بنابل
Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul

EPREUVE D'INFORMATIQUE
EXAMEN DU 1ER SEMESTRE
NIVEAU : 2EME ANNEE

Filières : SM, SP, ST | Durée : 2h | Nombre de pages : 4 | Date : 16/12/2017

AVERTISSEMENT

- L'épreuve est composée de 4 exercices, totalement indépendants.
- Vous pouvez toujours admettre le résultat des questions non faites pour faire les questions suivantes.

Exercice 1 : (3pts)

1. Citez deux avantages de la programmation orientée objet par rapport à la programmation procédurale. **[1]**

2. Citez un exemple d'un problème scientifique simple où sa résolution nécessite l'utilisation d'une structure de données avancée de type ***Pile*** ou bien de type ***File*** afin d'enregistrer vos données et vos résultats. Justifiez votre choix. **[1]**

3. Théoriquement, une structure de données avancée de type **Pile** peut avoir une capacité de stockage infinie. Selon vous, pratiquement, quelle est la contrainte majeure de ne pas avoir une pile à capacité infinie ? **[1]**

Exercice 2 : (3pts)

1. Qu'affichera ce programme ? Justifiez votre choix. **[1]**

```
class parent:
    def __init__(self, param):
        self.v1 = param

class enfant(parent):
    def __init__(self, param):
        self.v2 = param

obj = enfant(11)
print(obj.v1 + " " + obj.v2)
```

a) None None

b) None 11

c) 11 None

d) 11 11

e) AttributError

2. Qu'affichera ce programme ? Justifiez votre choix. **[1]**

```
class Compte:
    def __init__(self, id):
        self.id = id
        id = 100

cpte = Compte(123)
print(cpte.id)
```

a) SyntaxError, ce programme ne fonctionnera pas

b) None

c) 123

d) 100

e) Aucune de ces réponses

3. Quelles sont les réponses correctes à propos de l'extrait du code donné ? **[1]**

```
class A:
    def __init__(self, i = 0):
        self.i = i

class B(A):
    def __init__(self, j = 0):
        self.j = j

def main():
    b = B()
    print(b.i)
    print(b.j)

main()
```

a) La classe B hérite de A, mais le champ de données "i" dans A n'est pas hérité.

b) La classe B hérite de A, héritant ainsi automatiquement tous les attributs de données de A.

c) Lorsque vous créez un objet de B, vous devez passer un argument tel que B (5).

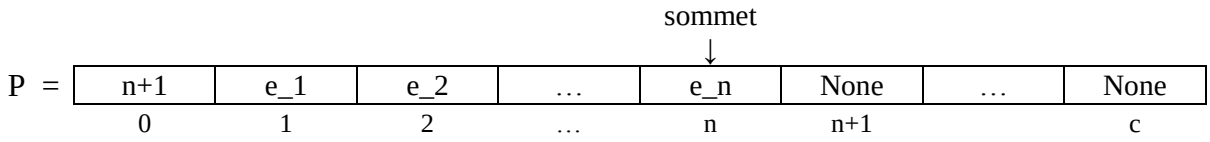
d) Le champ de données "j" n'est pas accessible par l'objet b.

Exercice 3 : (6pts)

Dans le cadre de cet exercice, on veut réaliser, à l'aide d'une liste, une pile à capacité finie permettant de stocker au plus **c** éléments:

- La première case (de rang **0**) contient l'indice du prochain élément à insérer dans la pile.
- Lorsque **P [0]** égale à **1**, la pile est considérée comme vide.
- Les **n** cases suivantes, d'indices de **1** à **n**, contiennent les éléments déjà insérés dans la pile.
- Le dernier élément inséré, d'indices de **n**, est appelé sommet de la pile.
- Les cases restantes, d'indices de **n+1** à **c**, n'appartiennent pas à la pile.
- À chaque fois qu'on insère un élément, on augmente **P [0]** d'une unité.
- Lorsque **P [0]** égale à **c+1**, la pile est considérée comme pleine.

Exemple : Une pile P, non pleine, contenant n éléments.



Travail à faire :

1. Ecrire une classe Python nommée « **MaPile »** implémentant une pile d'éléments. Définir le **constructeur** de cette classe qui initialisera une pile vide à une capacité maximale de **c** éléments. **[1]**
2. Définir une méthode **est_vide** permettant de retourner True si la pile est vide, sinon elle retourne False. **[0.5]**
3. Définir une méthode **est_pleine** permettant de retourner True si la pile est pleine, sinon elle retourne False. **[0.5]**
4. Définir une méthode **sommet** permettant de retourner la valeur du sommet de la pile. **[0.5]**
5. Définir une méthode **empile**r permettant d'ajouter un élément **e** à la pile. Cette méthode déclenche une exception de type IndexError si la pile est pleine. **[1.5]**
6. Définir une méthode **dépiler** permettant de retourner le sommet de la pile et de le mettre à jour à la bonne position. Cette méthode déclenche une exception de type IndexError si la pile est vide. **[1.5]**
7. Définir une méthode **__str__** permettant d'imprimer les éléments de la pile. **[0.5]**

Exemples d'utilisations de la classe MaPile :

```
>>> pile1 = MaPile(7)
>>> print(pile1)
[1, None, None, None, None, None, None, None]
>>> pile1.empiler('a')
>>> print(pile1)
[2, 'a', None, None, None, None, None, None]
>>> pile1.empiler('b')
>>> print(pile1)
[3, 'a', 'b', None, None, None, None, None]
>>> x = pile1.depiler()
>>> print(x)
b
>>> print(pile1)
[2, 'a', 'b', None, None, None, None, None]
```

Exercice 4 : (8 pts)

Programmons une classe **Vect2d** dont les objets modéliseront des vecteurs à deux composantes réelles dans un repère orthonormé (O ; I ; J), gradué avec la même unité (OI = OJ = 1 unité). Lorsque le mathématicien dit : « Soit le vecteur $\vec{u}(3,-2)$ », le programmeur Python dira : « u = Vect2d (3,-2) ».

1. Programmez le *constructeur*, chargé d'initialiser les attributs x et y de l'instance courante, qui se nomme self. Par défaut, x et y seront mis à 0. **[1]**
2. Si je veux additionner deux vecteurs, j'ai deux solutions. Ou bien je programme une fonction mathématique ***add(v1,v2)*** à l'extérieur de la classe. Ou bien, dans une optique de pure programmation par objets, je programme **add** comme méthode d'instance à l'intérieur de la classe. Adoptez cette seconde solution, pour laquelle le résultat de l'addition avec la méthode **add** sera un nouveau vecteur. **[1]**
3. Programmez une méthode ***mul_ext*** réalisant la multiplication $k\vec{u}$ d'un réel *k* par un vecteur $\vec{u}$. Le résultat sera un nouveau vecteur. **[1]**
4. Programmez une méthode ***zoom*** permettant à un vecteur de se multiplier par *k* et d'être ainsi définitivement modifié ! Faites bien la différence avec ***mul_ext***. **[1]**
5. Programmez une méthode ***prodscal*** demandant à un vecteur de retourner son produit scalaire avec un autre vecteur. **[1]**
6. En déduire une méthode ***norme*** demandant à un vecteur de retourner sa longueur. **[0.5]**
7. Programmez à l'extérieur de la classe une fonction ***add(v1,v2)*** retournant la somme vectorielle $\overrightarrow{v_1} + \overrightarrow{v_2}$. **[1]**

   Remarque : En principe il est impossible d'écrire u + v si u et v sont deux vecteurs. Mais en Python, des méthodes spéciales sont cachées sous les opérateurs. Si votre méthode **add** se nomme **__add__**, cela devient possible !

8. Afin de tester votre classe **Vect2d :** **[1.5]**
   a. Créez deux vecteurs $\vec{u}(3,-2)$ et $\vec{v}(4,1)$
   b. Créez un vecteur $\overrightarrow{w} = \vec{u} + \vec{v}$
   c. Vérifiez que $(\vec{u} + \overrightarrow{w}) \cdot \vec{v} = \vec{u} \cdot \vec{v} + \overrightarrow{w} \cdot \vec{v}$
   d. Vérifiez que $(5\,\overrightarrow{u}) \cdot \vec{v} = 5\,(\vec{u} \cdot \vec{v})$
   e. Affichez la valeur de l'angle α formé entre les deux vecteurs $\vec{u}$ et $\vec{v}$.
   Rappelons que :
   $\vec{u} \cdot \vec{v} = \|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos(\alpha)$
   En python la fonction **acos(*x*)** du module math retourne **cos^{-1} (*x*)**, l'arc cosinus de ***x***, en radians.
   Exemple : si *cos(x) = 1.0* alors *x = acos(1.0) = 0.0*

••• Fin de l'épreuve •••