POO : TD

Thème 1 - Structure de données

BAC TD : Sujet BAC Programmation Orientée Objet (POO)

1. Exercice 1 : (d'après Bac) - Locations de chambres.

Principaux thèmes abordés :

Structure de données (programmation objet) et langages et programmation (spécification).

La société LOCAVACANCES doit gérer la réservation de l'ensemble des chambres de ses gîtes. Chaque chambre d'un même complexe sera différenciée par son nom.

Pour cela, d'un point de vue informatique, on a créé deux classes : `Chambre` et `Gite` dont le code ci-dessous.

Script Python

```
class Chambre:
    def __init__(self, nom: str):
        self._nom = nom
        self._occupation = [False for i in range(365)]

        def get_nom(self):
            return self._nom

        def get_occupation(self):
            return self._occupation

        def reserver(self, date: int):
            self._occupation[date - 1] = True

class Gite:
    def __init__(self, nom: str):
        self._nom = nom
        self._chambres = []

    def __str__(self):
        n = len(self._chambres)
```

```
        if n == 0:
            return "L'hôtel " + self._nom + " n'a aucune chambre."
        else:
            return "L'hôtel " + self._nom + " a " + str(n) + " chambre(s)"

    def get_chambres(self):
        return self._chambres

    def get_nchambres(self):
        return [ch.get_nom() for ch in self._chambres]

    def ajouter_chambres(self, nom_ch : str):
        self._chambres.append(Chambre(nom_ch))

    def mystere(self, date):
        l_ch = []
        for ch in self._chambres :
            if ch.get_occupation()[date - 1] == False :
                l_ch.append(ch.get_nom())
        return(l_ch)
```

1.1. Partie A - Étude de la classe `Chambre` :

1. Lister les attributs en donnant leur type. Préciser s'ils sont modifiables dans la classe, en explicitant la méthode associée.

Réponse :

Les attributs sont en principe définis et initialisés dans la méthode `__init__`. On les désigne en faisant précéder leur nom du mot réservé `self`. Ici on a donc les attributs appelés `_nom` et `_occupation` .

On constate que lors de l'initialisation la valeur attribuée à `_nom` est le paramètre `nom` de la méthode `__init__` , dont le type est précisé comme étant `str`, c'est à dire chaîne de caractères.

`_occupation` est initialisé sous la forme d'une liste de valeurs booléennes (définie en compréhension)

La méthode `reserver` permet de modifier la valeur de l'attribut `_occupation`.

1. Écrire un `assert` dans la méthode `reserver` pour vérifier si le nombre `date` passé en paramètre est bien compris entre 1 et 365 (on ne gère pas les années bissextiles).

Réponse :

Modification de la méthode `reserver` :

Script Python

```
def reserver(self, date: int):
    assert date >=1 and date <=365
    self._occupation[date - 1] = True
```

ou

Script Python

```
def reserver(self, date: int):
    assert 1<= date <=365
    self._occupation[date - 1] = True
```

1. Écrire la méthode `annuler_reserver(self, date : int)` qui annule la réservation pour le jour `date`.

Réponse :

Script Python

```
def annuler_reserver(self, date:int):
    self._occupation[date - 1] = False
```

1.2. Partie B - Étude de la classe `Gite` :

Le gîte « BonneNuit » a 5 chambres dénommées :

'Ch1', 'Ch2', 'Ch3', 'Ch4', 'Ch5'

On définit l'objet `giteBN` par l'instruction : `giteBN = Gite("BonneNuit")`.

1. Méthode `ajouter_chambres()`

Écrire l'instruction Python pour ajouter `'Ch1'` à l'objet `giteBN`.

Réponse :

Ajout d'une chambre à `giteBN` en appelant sa méthode `ajouter_chambre` avec en paramètre le nom de la chambre :

Script Python

```
giteBN.ajouter_chambres('Ch1')
```

Dans les questions suivantes 2, 3 et 4, on considère que l'objet `giteBN` contient toutes les chambres du gite « BonneNuit ».

1. La méthode `ajouter_chambres` permet d'enregistrer une nouvelle chambre, mais elle ne teste pas si le nom de cette chambre existe déjà.
   Modifier la méthode pour éviter cet éventuel doublon.

Réponse :

Les chambres sont enregistrées dans l'attribut `_chambres` de l'objet gite. Il faut inspecter tous les noms de chambre dans cette liste pour savoir si le nouveau nom proposé existe déjà.
La méthode `get_nchambres` renvoie la liste des noms de toutes les chambres du gite. On peut donc faire :

Script Python

```
def ajouter_chambres(self, nom_ch : str):
    if nom_ch not in self.get_nchambres():
        self._chambres.append(Chambre(nom_ch))
```

ou bien, sans utiliser `get_nchambres`, une démarche classique où l'on parcourt la liste de chambres en positionnant un indicateur booléen (`nom_nouveau`) pour signaler éventuellement que le nom choisi existait déjà :

Script Python

```
def ajouter_chambres(self, nom_ch : str):
    nom_nouveau = True
    for chambre in self._chambres:
        if chambre.get_nom() == nom_ch:
            nom_nouveau = False
    if nom_nouveau:
        self._chambres.append(Chambre(nom_ch))
```

1. Étude des méthodes : `get_chambres()` et `get_nchambres()`.

a. Parmi les 4 propositions ci-dessous, quel est le type renvoyé par l'instruction Python : `giteBN.get_chambres()` .

- String
- Objet Chambre
- Tableau de String
- Tableau d'objets Chambre

Réponse :

Cette méthode renvoie la valeur de l'attribut `_chambres` , qui est une liste. La méthode `ajouter_chambres` de la classe gite modifie cette liste en lui ajoutant un nouvel objet Chambre (créé par l'appel de `Chambre(nom_ch)`) .

La réponse est donc "tableau d'objets Chambre" (une liste Python est un tableau dynamique)

b. Qu'affiche la suite d'instructions suivante ?

Script Python

```
ch = giteBN.get_chambres()[1]
print(ch.get_nom())
```

Réponse :

La première ligne récupère le second élément de la liste des chambres et la seconde affiche le nom de cette chambre.

c. Quelle différence existe-t-il entre les deux méthodes `get_nchambres()` et `get_chambres()` ?

Réponse :

La méthode `get_nchambres` renvoie une liste de chaines de caractères qui sont les noms des chambres, et `get_chambres` renvoie une liste d'objets Chambre.

1. Les chambres `'Ch1', 'Ch3', Ch5'` sont réservées pour tout le mois de Janvier 2021.

a. Que va renvoyer l'instruction `giteBN.mystere(3)` ?

Réponse :

La méhode `mystere` renvoie une liste à laquelle ont été ajoutés les noms des chambres du gite à condition qu'elles ne soient pas réservées à la date passée en paramètre.

Donc `giteBN.mystere(3)` va renvoyer la liste des noms des chambres du gite "BonneNuit" disponibles le 4 janvier. Cette liste ne contiendra pas les noms `'Ch1', 'Ch3', Ch5'` puisque ces chambres sont réservées en Janvier. Elle contiendra peut-être `'Ch2'` et `'Ch4'` dont on ne sait pas si elles sont réservées le 4 janvier.

b. Dans la méthode `mystere` de la classe `Gite` , quel est le type des variables en paramètre et en sortie ? Quelles sont les méthodes ou attributs dont cette méthode a besoin ?

Réponse :

La méthode `mystere` prend comme paramètre `date`, qui est un entier (`int`).

Elle retourne une liste Python.

Elle a besoin de l'attribut `_chambres` des la classe `Gite` (ainsi que de la méthode `get_occupation` et `get_nom` de la classe `Chambre` , et la méthode `append` de la classe `list`. La question n'est pas très précise quand à ce qui est attendu)

2. Exercice 2 : (d'après Bac) - Pots de yaourts "

Principaux thèmes abordés :

structure de données (programmation objet) et langages et programmation (spécification).

Une entreprise fabrique des yaourts qui peuvent être soit nature (sans arôme), soit aromatisés (fraise, abricot ou vanille).

Pour pouvoir traiter informatiquement les spécificités de ce produit, on va donc créer une classe Yaourt qui possèdera un certain nombre d'attributs :

- Son genre : nature ou aromatisé
- Son arôme : fraise, abricot, vanille ou aucun
- Sa date de durabilité minimale (DDM) exprimée par un entier compris entre 1 et 365 (on ne gère pas les années bissextiles). Par exemple, si la DDM est égale à 15, la date de durabilité minimale est le 15 janvier.

On va créer également des méthodes permettant d'interagir avec l'objet Yaourt pour attribuer un arôme ou récupérer un genre par exemple. On peut représenter cette classe par le tableau de spécifications ci-dessous :

Yaourt

ATTRIBUTS	METHODES
genre	construire(arome,duree)
arome	obtenir_arome()
duree	obtenir_genre()
	obtenir_duree()
	attribuer_arome(arome)
	attribuer_duree(duree)
	attribuer_genre(arome)

1. Code partiel de la classe Yaourt, à compléter aux endroits indiqués en suivant les consignes des questions suivantes :

Script Python

```
class Yaourt:
    """ Classe définissant un yaourt caractérisé par :
        - son arome
        - son genre
        - sa durée de durabilité minimale"""

    def __init__(self,arome,duree):
    # **** Assertions : à compléter suivant les indications de la  question 1.a.
```

```
        self.__arome = arome
        self.__duree = duree
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'

    # **** Méthode get_arome(self) à compléter suivant les indications de la question 1.c.

    def get_duree(self):
        return self.__duree

    def get_genre(self):
        return self.__genre


    def set_duree(self,duree):
        # **** Mutateur de durée
        if duree > 0 :
            self.__duree = duree

    # **** Mutateur d'arôme set_arome(self,arome) - à compléter suivant les indications de la
question 2.

    def __set_genre(self,arome):
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'
```

1.a. Quelles sont les assertions à prévoir pour vérifier que l'arôme et la durée correspondent bien à des valeurs acceptables? Il faudra aussi expliciter les commentaires qui seront renvoyés.
Pour rappel :

- L'arôme doit prendre comme valeur 'fraise', 'abricot', 'vanille' ou 'aucun'.
- Sa date de durabilité minimale (DDM) est une valeur positive.

Réponse :

Modification de la méthode `__init__` de la classe Yaourt :

Script Python

```
    def __init__(self,arome,duree):
        assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
        assert duree >= 0, "La DDM doit être une valeur positive"
        self.__arome = arome
        self.__duree = duree
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'
```

1.b. Pour créer un yaourt, on exécutera la commande suivante :

Script Python

```
mon_yaourt = Yaourt('fraise',24)
```

Quelle valeur sera affectée à l'attribut `genre` associé à `mon_yaourt` ?

Réponse :

L'attribut `genre` aura comme valeur 'aromatise' puisque l'arome du yaourt n'est pas ègal à 'aucun'.

1.c. Écrire en Python une fonction `get_arome(self)`, renvoyant l'arôme du yaourt créé.

Réponse :

Méthode `get_arome` à insérer dans le code de la classe Yaourt :

Script Python

```
def get_arome(self):
    return self.__arome
```

1. On appelle mutateur une méthode permettant de modifier un ou plusieurs attributs d'un objet.
   Ecrire en Python le mutateur `set_arome(self,arome)` permettant de modifier l'arôme du yaourt.
   On veillera à garder une cohérence entre l'arôme et le genre.

Réponse :

Méthode `set_arome` à insérer dans le code de la classe Yaourt :

Script Python

```
def set_arome(self, arome):
    assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
    self.__arome=arome
    if arome == 'aucun':
        self.__genre = 'nature'
    else:
        self.__genre = 'aromatise'
```

On peut aussi utiliser la méthode `set_genre` déjà définie pour garder une cohérence entre l'arôme et le genre.:

Script Python

```
def set_arome(self, arome):
    assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
    self.__arome=arome
    self.__set_genre(arome)
```

1. On veut créer une pile contenant le stock de yaourts. Pour cela il faut tout d'abord créer une pile vide :

Script Python

```
def creer_pile():
    pile = [ ]
    return pile
```

3.a. Créer une fonction `empiler( p, yaourt:Yaourt)` qui renvoie la pile `p` après avoir ajouté un objet de type Yaourt au sommet de la pile.

Réponse :

Fonction `empiler` (on suppose que le sommet de la pile est la fin de la liste):

Script Python

```
def empiler(p, yaourt):
    p.append(yaourt)
```

3.b. Créer une fonction `depiler(p)` qui renvoie l'objet à dépiler.

Réponse :

Fonction `dépiler` :

Script Python

```
def depiler(p):
    return p.pop(-1)
```

3.c. Créer une fonction `est_vide(p)` qui renvoie `True` si la pile est vide et `False` sinon.

Réponse :

Fonction `est_vide` :

Script Python

```
def est_vide(p):
    return len(p) == 0
```

ou bien, en plus explicite :

Script Python

```
def est_vide(p):
    if len(p) == 0:
        return True
    else:
        return False
```

3.d. Qu'affiche le bloc de commandes ci-dessous ?

Script Python

```
mon_yaourt1 = Yaourt('aucun',18)
mon_yaourt2 = Yaourt('fraise',24)
ma_pile = creer_pile()
empiler(ma_pile, mon_yaourt1)
empiler(ma_pile, mon_yaourt2)
print(depiler(ma_pile).get_duree())
print(est_vide(ma_pile))
```

Réponse :

Le sujet d'origine contient un espace entre mon_yaourt et 2 à la ligne 5, ce qui fait que l'exécution du code tel quel génèrerait une erreur. Il s'agit cependant sans doute d'une coquille.

Pour le code corrigé donné ci-dessus :
Les deux premières lignes créent deux objets Yaourt. La troisième crée une pile, où les deux Yaourts sont empilés aux deux lignes suivantes. A la ligne 6, la commande dépile le dernier Yaourt empilé, `mon_yaourt2`, et affiche sa DDN, c'est à dire 24. Enfin la dernière ligne affiche `False` puisqu'à ce stade la pile contient encore `mon_yaourt1`, donc n'est pas vide.

Script Python

```
class Yaourt:
    """ Classe définissant un yaourt caractérisé par :
        - son arome
        - son genre
        - sa durée de durabilité minimale"""

    def __init__(self,arome,duree):
        assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
        assert duree >= 0, "La DDM doit être une valeur positive"
        self.__arome = arome
        self.__duree = duree
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'

    def get_arome(self):
        return self.__arome

    def get_duree(self):
        return self.__duree

    def get_genre(self):
        return self.__genre

    def set_duree(self,duree):
        # **** Mutateur de durée
        if duree > 0 :
            self.__duree = duree

    def set_arome(self, arome):
        assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
        self.__arome=arome
        self.__set_genre(arome)

    def __set_genre(self,arome):
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'

def creer_pile():
    pile = [ ]
    return pile

def empiler(p, yaourt):
    p.append(yaourt)

def depiler(p):
    return p.pop(-1)
```

```
def est_vide(p):
    return len(p) == 0

mon_yaourt1 = Yaourt('aucun',18)
mon_yaourt2 = Yaourt('fraise',24)
ma_pile = creer_pile()
empiler(ma_pile, mon_yaourt1)
empiler(ma_pile, mon_yaourt2)
print(depiler(ma_pile).get_duree())
print(est_vide(ma_pile))
```

24
False