#### 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 :

return self. nom

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 cidessous.

# class Chambre: def \_\_init\_\_(self, nom: str): self.\_nom = nom self.\_occupation = [False for i in range(365)] def get nom(self):

```
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."
       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):
     {\color{red} 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.

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).



1. Écrire la méthode annuler\_reserver(self, date : int) qui annule la réservation pour le jour date .



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

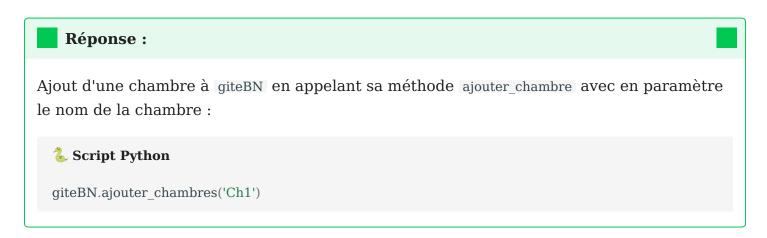
Le gîte « BonneNuit » a 5 chambres dénommées :

'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.



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.

Les chambres sont enregistrées dans l'attribut <u>chambres</u> 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 <code>get\_nchambres</code> 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 <code>get\_nchambres</code>, une démarche classique où l'on parcourt la liste de chambres en positionnant un indicateur booléen (<code>nom\_nouveau</code>) pour signaler éventuellement que le nom choisi existait déjà :

```
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



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) ?

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 :



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

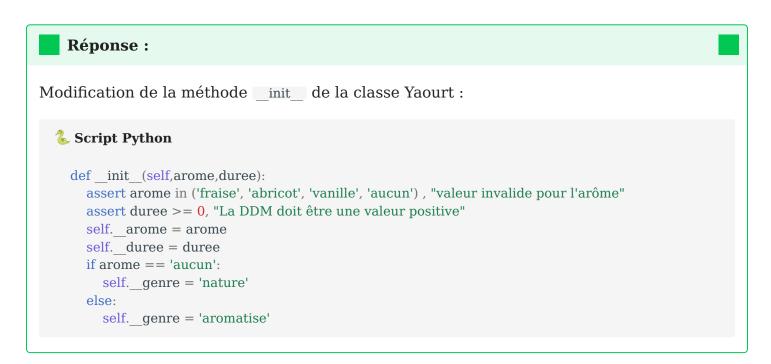
# **** Mutateur d'arôme set_arome(self,arome) - à compléter suivant les indications de la question 2.

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.



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

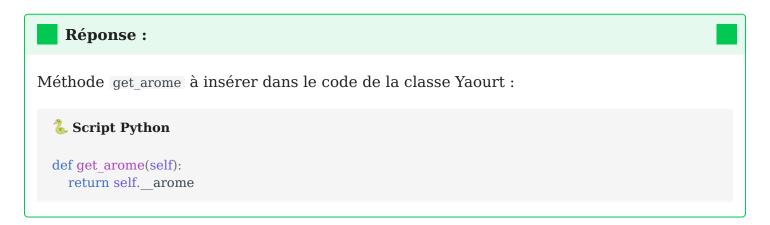
```
Script Python
mon_yaourt = Yaourt('fraise',24)
```

Quelle valeur sera affectée à l'attribut genre associé à mon\_yaourt ?



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.



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.

Méthode set arome à insérer dans le code de la classe Yaourt :

```
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.:

```
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.

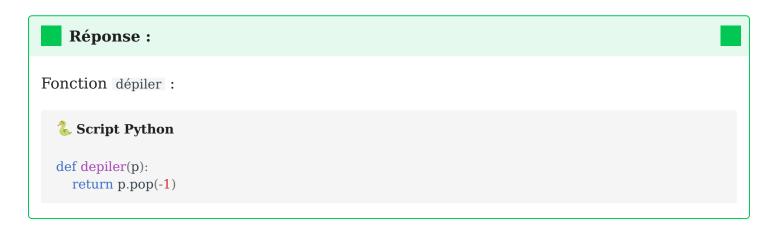
# Réponse :

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

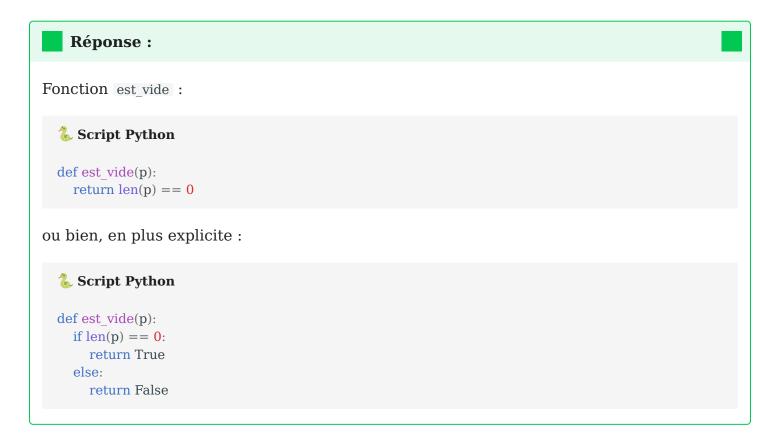
Script Python

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

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



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



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

```
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))
```

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