Exercice 2 (6 points)

Cet exercice porte sur la gestion des bugs, l'algorithmique, les structures de données et la programmation orientée objet.

Partie A

Un jour, Bob s'apprête à manger un collier de bonbons, et se pose la question suivante : « *Si je mange un bonbon sur trois, encore et encore jusqu'à ce qu'il n'en reste qu'un seul, quel sera le dernier bonbon restant ?* »

Figure 1. Collier de bonbons

Pour un collier ayant `5` bonbons, il décide de les numéroter de `0` à `4`. Il commence par manger le bonbon d'indice `0`, se décale de trois bonbons et mange ensuite celui d'indice `3`. En répétant la démarche, il mange ensuite le bonbon d'indice `2` et enfin celui d'indice `4`.

Les indices des bonbons mangés sont donc, dans l'ordre, `0`, `3`, `2` et `4`. Le bonbon restant est celui d'indice `1`.

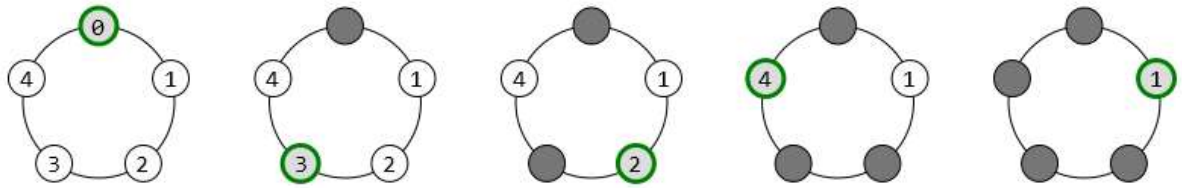

Figure 2. Les étapes pour un collier de `5` bonbons

1. Donner les indices dans l'ordre dans lequel les bonbons sont mangés dans le cas où le collier possède initialement `8` bonbons et l'indice du bonbon restant.

Afin de répondre à la question dans un cadre général, Bob décide de formaliser le problème. Il considère un collier de `n` bonbons numérotés de `0` à `n - 1`, où `n` est un entier strictement positif.

Bob vient d'étudier en classe les valeurs booléennes. Il se dit qu'il peut représenter avec Python le collier par une liste `collier` telle que, pour toute valeur entière de `i` comprise entre `0` et `n - 1`, la valeur booléenne `collier[i]` vaut `True` si le bonbon d'indice `i` du `collier` est encore présent, et vaut `False` si le bonbon d'indice `i` du `collier` a été mangé.

Dès lors, il envisage l'algorithme suivant :

- on initialise une liste `collier` représentant `n` bonbons qui n'ont pas encore été mangés ;
- on commence par manger le bonbon à l'indice 0 ;
- tant qu'il reste des bonbons à manger :
  - on détermine l'indice du prochain bonbon à manger dans la liste `collier ;`
  - on mange le bonbon à cet indice ;
- on renvoie l'indice du dernier bonbon mangé.

Afin de créer la liste `collier` décrite ci-dessus, Bob saisit dans la console l'instruction

```
collier = [true for i in range(8)]
```

Il obtient alors le message d'erreur suivant :

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

2. Expliquer ce qu'est une erreur de type `NameError` et comment la corriger dans l'instruction proposée.

Bob écrit ensuite le code d'une fonction `dernier` qui prend en paramètre le nombre de bonbons `n` et renvoie l'indice du dernier bonbon restant. On fournit ci-après une partie du code de la fonction `dernier`.

```
 1 def dernier(n):
 2     collier = ...
 3     indice = 0
 4     collier[indice] = False
 5     for etape in range(n-1):
 6         nb_bonbons_vus = 0
 7         while nb_bonbons_vus ...:
 8             indice += 1
 9             if ...:
10                 indice = 0
11             if ...:
12                 nb_bonbons_vus += 1
13         collier[indice] = ...
14     return indice
```

3. Recopier et compléter les lignes 2, 7, 9, 11 et 13 du code de la fonction `dernier`.

Partie B

Bob se dit qu'une structure de file lui permettrait de résoudre astucieusement le problème des bonbons.

On considère la classe `File` dont on fournit ci-après l'interface.

```
 1 class File:
 2     """Classe définissant une structure de file"""
 3
 4     def __init__(self):
 5         """Initialise une file vide"""
 6
 7     def est_vide(self):
 8         """Renvoie le booléen indiquant
 9          si la file est vide"""
10
11     def enfile(self, x):
12         """Place x à la queue de la file"""
13
14     def defile(self):
15         """Retire et renvoie l'élément placé à la
16         tête de la file
17         Provoque une erreur si la file est vide
18         """
19
20     def affiche(self):
21         """Affiche la file"""
```

Le code Python ci-après montre un exemple d'utilisation de la classe `File`.

```
>>> f = File()
>>> f.enfile(0)
>>> f.enfile(1)
>>> f.affiche()
(Tête) 0 1 (Queue)
```

L'acronyme LIFO signifie « Last In First Out » à savoir « Dernier entré, premier sorti ». L'acronyme FIFO signifie « First In First Out » à savoir « Premier entré, premier sorti ».

4. Donner l'acronyme le plus adapté à la structure de donnée `File`.

5. Déterminer l'affichage réalisé lors de l'exécution des instructions ci-après.

```
>>> f = File()
>>> for x in [0, 1, 2, 3, 4]:
>>>     f.enfile(x)
>>> f.defile()
>>> f.enfile(f.defile())
>>> f.enfile(f.defile())
>>> f.affiche()
```

6. Écrire le code de la fonction `dernier_file` qui prend en paramètre le nombre de bonbons `n` et renvoie l'indice du dernier bonbon restant.

Partie C

Bob souhaite utiliser la structure de données « *liste doublement chaînée* ». Une telle liste est composée de *maillons* contenant chacun trois informations :

- une valeur ;
- un prédécesseur et un successeur qui sont tous deux des maillons.

Cette structure se prête bien au problème des bonbons : dans un collier, un bonbon est précédé et suivi par d'autres bonbons. Le successeur du dernier bonbon est le premier et le prédécesseur du premier, le dernier.

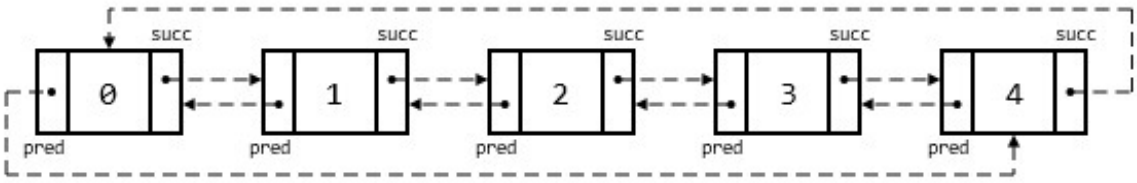

Figure 3. Collier de cinq `Bonbon`

Bob crée la classe `Bonbon` ci-après :

```
1 class Bonbon:
2     def __init__(self, valeur):
3         self.pred = None  # prédécesseur de ce bonbon
4         self.valeur = valeur
5         self.succ = None  # successeur de ce bonbon
```

7. Donner le terme correspondant aux variables `pred`, `valeur` et `succ` dans le vocabulaire de la programmation orientée objet.

Les instructions ci-dessous permettent de représenter un collier de trois bonbons de valeurs `0`, `1` et `2`.

```
>>> zero = Bonbon(0)
>>> un = Bonbon(1)
>>> deux = Bonbon(2)

>>> zero.succ = un
>>> un.pred = zero

>>> un.succ = deux
>>> deux.pred = un

>>> deux.succ = zero
>>> zero.pred = deux

>>> a = zero.succ.valeur
>>> b = un.succ.succ.pred.valeur
```

8. Déterminer les valeurs des variables `a` et `b` après l'exécution de ces instructions.

La fonction `creer_collier` prend en paramètre un entier `n` strictement positif représentant la taille d'un collier et renvoie un objet de type `Bonbon` représentant le premier bonbon (de valeur `0`) du collier.

On prendra soin de faire se succéder et précéder les différents bonbons ainsi que de « refermer » le collier en liant le dernier bonbon au premier.

9. Recopier et compléter les lignes 5, 6, 7, 9 et 10 du code de la fonction `creer_collier`, donné ci-après.

```
 1 def creer_collier(n):
 2     premier = Bonbon(0)
 3     actuel = premier
 4     for i in range(1, n):
 5         nouveau = Bonbon(...)
 6         actuel.succ = ...
 7         ...
 8         actuel = nouveau
 9     ...
10     ...
11     return premier
```

On considère le code Python suivant.

```
>>> bonbon = Bonbon(3)
>>> bonbon.pred = bonbon
>>> bonbon.succ = bonbon
```

À l'issue de l'exécution de ce code, on obtient la liste doublement chaînée représentée ci-dessous.

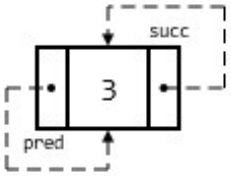

Figure 4. Un collier d'un seul `Bonbon`

10. On considère le code Python suivant.

    ```
    >>> premier = creer_collier(4)
    >>> premier.pred.succ = premier.succ
    >>> premier.succ.pred = premier.pred
    >>> bonbon = premier.succ
    ```

    Dessiner une représentation du « collier » dont le premier élément est l'objet `bonbon` obtenu à l'issue de l'exécution du code Python ci-dessus.

11. Dans le cas où il ne reste qu'un bonbon, donner l'expression qui s'évalue à `True`, parmi les quatre propositions ci-dessous :

    - Proposition A : `valeur.succ == valeur.bonbon`
    - Proposition B : `pred == succ`
    - Proposition C : `bonbon.valeur == bonbon.succ.valeur`
    - Proposition D : `bonbon.valeur == succ.valeur`

12. Recopier et compléter les lignes 3, 4, 5 et 6 du code de la fonction `dernier_chaine`, donné ci-après, qui prend en paramètre le nombre de bonbons `n` et renvoie la valeur du dernier bonbon restant.

    ```
    1 def dernier_chaine(n):
    2     bonbon = creer_collier(n)
    3     while ... != ...:
    4         bonbon.pred.succ = ...
    5         ... = bonbon.pred
    6         bonbon = ... # décalage de 3 bonbons
    7     return bonbon.valeur
    ```