3.2 Ressources

On s'intéresse à la programmation d'une structure abstraite permettant de stocker un ensemble de ressources. Chaque ressource est unique et identifiable par un identifiant entier unique compris entre 0 et le nombre total de ressources -1. On ne stocke dans cet exercice que ces identifiants (aussi appelés *indices*).

On pourrait utiliser un **set** de python mais cette structure nécessite de stocker toutes les ressources contenues dans le set, ce qui peut être lent dans le cas où le nombre de ressources disponibles est particulièrement élevé.

À l'inverse, on se propose de programmer nous-même une structure, en réalisant une compression des données stockées par plages contigües. On utilisera pour ce faire un tableau d'intervalles. Chaque intervalle sera lui-même représenté par un tableau de deux éléments : l'indice de début et l'indice suivant l'indice de fin.

Par exemple, l'ensemble {0,1,3,5,6,9,10,11,12} sera représenté par 4 intervalles : `[0,2], [3,4], [5,7], [9,13]`. Inversemment les intervalles `[1,3], [5,7], [8,10]` représentent l'ensemble {1,2,5,6,8,9}.

On implémentera donc une classe `Ressources` dont le constructeur est donné ci-dessous :

```
class Ressources:
    """
    On stocke une liste de ressources, compressee par plages contigues.
    """
    def __init__(self, nombre_ressources, intervalles=None):
        # requiert : si intervalles is not None, alors :
        #            - les intervalles sont non vides
        #            - les intervalles sont non contigus
        #            - les intervalles sont tries par indices croissants
        #            - intervalles[-1][1] <= nombre_ressources
        self.nombre_ressources = nombre_ressources
        if intervalles is not None:
            self.intervalles = intervalles
        else:
            self.intervalles = [[0, nombre_ressources]]
```

On vous demande en particulier d'implémenter les méthodes suivantes :

- `disponible(self, indice)` : renvoie si l'indice donné est disponible dans l'ensemble ;
- `reserve(self, ressources_demandees)` : diminue les ressources de `self` du nombre de ressources demandées (en enlevant les premières). Renvoie un nouvel object `Ressources` contenant les identifiants des ressources enlevées.
- `retourne(self, ressources_enlevees)` : remet dans `self` les ressources données, enlevées précedemment.
- `__str__(self)` : renvoie une chaine délimitée par des barres verticales et contenant un caractère '.' pour chaque ressource absente de `self` et un caractère 'x' pour chaque ressource présente dans `self`. Par exemple, les ressources [0,1], [2,4], [6,7] (sur 10 ressources au total), correspondant á l'ensemble {0,2,3,6}, sont visualisées par `'|x.xx..x...|'`.

Vous pouvez compléter le fichier ressources.py.