TP 4

1 Retour sur la classe `Compteur`

Remplacer la méthode `elements(self)` de la classe `Compteur` du module `compteur` par la propriété `elements` en utilisant le décorateur.

2 Module `code_binaire`

Développer le module `code_binaire` qui propose un type énuméré `Bit` et une classe `CodeBinaire`. L'appel à la fonction **repr** doit retourner une chaine évaluable. L'appel à la fonction **str** doit retourner `"0"` ou `"1"`.

2.1 Énuméré `Bit`

Le type énuméré `Bit` qui propose deux valeurs `BIT_0` et `BIT_1`.

2.2 Classe `CodeBinaire`

La classe `CodeBinaire` propose les méthodes suivantes :

— `__init__(self, bit, *bits)` qui permet d'initialiser un code binaire avec au moins un bit ;
— `ajouter(self, bit)` qui permet d'ajouter un bit (à la fin).
Elle propose aussi la propriété `bits` qui permet d'obtenir un tuple des bits du code binaire.
De plus on peut :
— obtenir la longueur d'un code binaire à l'aide de la fonction **len** ;
— concatener (opérateur +) deux codes binaires ;
— obtenir ou modifier une partie du code binaire grâce à la notation `[ ]`. Attention si l'opérande est un indice on obtient ou on fixe un bit, si c'est un *slice* on obtient un code binaire ou on fixe un code binaire ou une séquence de bits ;
— supprimer un bit ou un ensemble de bits grâce à la fonction **del** ;
— itérer sur les bits du code binaire ;
— tester l'égalité (opérateur ==) de deux codes binaires ;
— la représentation informelle sera une suite de 0 et de 1.
Voici quelques exemples d'utilisation de cette classe :

```
In [1]: from code_binaire import Bit, CodeBinaire

In [2]: c=CodeBinaire(Bit.BIT_0, Bit.BIT_1)

In [3]: c
Out[3]: CodeBinaire(Bit.BIT_0, Bit.BIT_1)
```

```
In [4]: c.ajouter(Bit.BIT_0)

In [5]: c
Out[5]: CodeBinaire(Bit.BIT_0, Bit.BIT_1, Bit.BIT_0)

In [6]: len(c)
Out[6]: 3

In [7]: c[1:3]
Out[7]: CodeBinaire(Bit.BIT_1, Bit.BIT_0)

In [8]: c[0]
Out[8]: Bit.BIT_0

In [9]: c[0:2]=[Bit.BIT_1, Bit.BIT_1, Bit.BIT_0]

In [10]: c
Out[10]: CodeBinaire(Bit.BIT_1, Bit.BIT_1, Bit.BIT_0, Bit.BIT_0)

In [11]: c[0:2]=CodeBinaire(Bit.BIT_0, Bit.BIT_1)

In [12]: c
Out[12]: CodeBinaire(Bit.BIT_0, Bit.BIT_1, Bit.BIT_0, Bit.BIT_0)


In [13]: del(c[0])

In [14]: c
Out[14]: CodeBinaire(Bit.BIT_1, Bit.BIT_0, Bit.BIT_0)

In [15]: for b in c:
   ....:     print(b)
   ....:
Bit.BIT_1
Bit.BIT_0
Bit.BIT_0

In [16]: print(c)
100

In [17]: c == CodeBinaire(Bit.BIT_1, Bit.BIT_0, Bit.BIT_0)
Out[17]: True
```