3 TP 3

Exercice 26. Une première classe
Dans l'interpréteur Python, exécutez :

```
>>> class Citron:
...     """ Une classe qui représente un citron """
...     couleur = 'jaune'
...
>>> help(Citron)
>>> dir(Citron)
>>> Citron.__dict__
>>> citron1 = Citron()
>>> type(citron1)
>>> isinstance(citron1, Citron)
>>> dir(citron1)
>>> citron1.couleur='rouge'
>>> citron1.couleur
>>> citron2= Citron()
>>> citron2.couleur
>>> citron2.__dict__
>>> citron1.__dict__
>>> Citron.couleur
```

Exercice 27. Créer un script contenant :

```
class Citron:
    couleur = 'jaune' # un attribut de classe

    def __init__(self,nom='citron',taille=100):
        self.taille = taille  # un premier d'attribut d'instance
        self.nom = nom        # un second attribut d'instance

    def multiplie(self,n):
        print((self.nom + ' ')*n)

    def __str__(self):
        return f"Mon nom est '{self.nom}', \
je suis {self.couleur} et je mesure {self.taille} mm."


citron1 = Citron()
citron2 = Citron('petit citron',1)
```

```
print(citron1)
print(str(citron1))

citron1.multiplie(3)
citron2.multiplie(2)

Citron.multiplie(citron1,3)

citron2.couleur = 'rouge'
print(citron2)
print(citron1)
```

Exercice 28. Le script suivant définit une classe `Personne` possédant un attribut `nom` et une classe dérivée `Enseignant` possédant en plus un attribut `section` .

```
class Personne:
    def __init__(self,nom):
        self.nom = nom
    def __str__(self):
        return self.nom

class Enseignant(Personne):
    def __init__(self, nom, section):
        Personne.__init__(self,nom)
        self.section = section
    def __str__(self):
        return Personne.__str__(self) + ' ' + str(self.section)

p = Personne('Machin')
print(p)
en = Enseignant('Truc', 26)
print(en)
```

1. Définir une classe `Etudiant` dérivée de `Personne`, munie d'un attribut `numero`. Redéfinir aussi les méthodes `__init__` et `__str__`.
2. Définir une classe `EnseignantEtudiant` qui dérive de `Enseignant` et `Etudiant`. La syntaxe pour signifier que cette classe possède ces deux classes mères est la suivante :

   ```
   class EnseignantEtudiant(Etudiant, Enseignant):
   ```

   Redéfinir aussi les méthodes `__init__` et `__str__`.

Exercice 29. Redéfinition des opérateurs

```
class Chaine:
    def __init__(self,value):
        self.s = str(value)

    def __str__(self):
        return self.s

    def __add__(self,other):
        """ renvoie self + other """
        return Chaine(self.s+other.s)

    def __mul__(self,other):
        """ renvoie self * other """
        c=''
        for x in zip(self.s,other.s):
            c+= x[0] + x[1]
        return Chaine(c)

if __name__ == "__main__":
    print(Chaine('abc') * Chaine('efgh'))
    print(Chaine('abc') + Chaine('efgh'))
```

Exercice 30. Concevoir une classe `Matrice` permettant de représenter une matrice dont les élements peuvent être additionnés (opérateur +) et multipliés (opérateur *).
Pour cette classe `Matrice`, définir le comportement de l'opérateur +, ainsi que celui de l'opérateur @ correspondant à la multiplication matricielle. L'opérateur @ est défini par la méthode

```
__matmul__(self, other)
```

qui a vocation à renvoyer `self @ other`.

Effectuer un test avec des matrices de nombres et aussi des matrices de `Chaine` (cf exercice 29).