1) Rappel du vocabulaire de la POO

- Le type de données avec ses caractéristiques et ses actions possibles s'appelle classe.
- Les caractéristiques (ou variables) de la classe s'appellent les attributs.
- Les actions possibles à effectuer avec la classe s'appellent les méthodes.
- La classe définit donc les attributs et les actions possibles sur ces attributs, les méthodes.
- Un objet du type de la classe s'appelle une instance de la classe et la création d'un objet d'une classe s'appelle une instanciation de cette classe.
- Lorsqu'on définit les attributs d'un objet de la classe, on parle d'instanciation.
- On dit que les attributs et les méthodes sont encapsulés dans la classe.

On peut afficher les méthodes associées à un objet avec la fonction dir(objet) :

```
# Dans la console PYTHON
>>> liste=[1,5,2]
>>> dir(liste)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__','__doc__','__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__','__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__','__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__','__setattr__','__setitem__', '__sizeof__', '__str__',
'__subclasshook__','append', 'clear', 'copy','count', 'extend', 'index', 'insert',
'pop','remove', 'reverse', 'sort']
```

On retrouve les méthodes connues concernant les listes : sort(), count(), append(), pop() ...
Les autres méthodes encadrées par **les underscores sont spéciales**.

2) Un jeu de cartes.

L'objectif de ce projet est de réaliser un jeu de bataille comprenant 4 classes dans 4 fichiers distincts avec un programme principal utilisant ces classes.

a) Production à rendre :

Les fichiers python pour chaque classe contenant :

- La définition de la classe (docstring),
- Des commentaires adaptés à vos choix
- La fonction de test de la classe.

b) Objectif

Exemple de sortie en cours d'exécution :

```
main de  paul : 9  Roi  8  Dame  9  # pour analyse
main de lea : 10  Roi  7  As  Dame  # pour analyse
Tour1: cartes de paul Dame CARREAU et lea As TREFLE c_cachées: 9 Dame
    lea : As gagne sur paul : Dame
Tour2: cartes de paul Roi CARREAU et lea Roi TREFLE c_cachées: 8 7
    paul : Roi à egalité avec lea : Roi
Tour3: cartes de paul 9 COEUR et lea 9 PIQUE c_cachées: 9 10
    paul : 9 à egalité avec lea : 9
Le gagnant est lea
```

c) Règle

Pour limiter le nombre de tours, cette version à tour double se fait en posant une carte cachée inconnue puis une carte lisible dessus.
La carte la plus grande valeur désigne le gagnant.
Le gagnant du tour ramasse les tas et les placent sous sa pile de cartes en position retournée.
En cas d'égalité, les joueurs font un autre tour sur les tas existants.
Dans le cas où un joueur n'a plus qu'une carte, il joue le tour de la dernière chance
Le gagnant est le joueur qui possède toutes les cartes

Rappel : commentez vos programmes !

3) La classe Carte (fichier carte.py).

Variables globales

```
couleurs = ('CARREAU', 'COEUR', 'TREFLE', 'PIQUE')
noms = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Valet', 'Dame', 'Roi','As']
valeurs = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10,
'Valet': 11, 'Dame': 12, 'Roi': 13, 'As':14}
```

a) La classe Carte

```
class Carte:
      def __init__(self, nom, couleur):
            # Affectation des attributs nom et couleur avec contrôle.
            self.nom = nom
            self.couleur = couleur
            self.valeur = ...
      ### Définition des méthodes d'instances avec contrôle (getter)####
      def getNom(self): # Accesseur de l'attribut,  renvoi le nom
      ...
      def getCouleur(self): # Accesseur de l'attribut , renvoi la couleur
      ...
      def getValeur(self): # Accesseur de l'attribut
      ...
      def egalite(self, carte):
      ''' Renvoie True si les cartes ont même valeur, False sinon
      carte: Objet de type Carte '''
      def estSuperieureA(self, carte):
      ''' Renvoie True si la valeur de self est supérieure à celle de carte,
      False sinon.  :carte: Objet de type Carte '''
      def estInferieureeA(self, carte):
      ''' Renvoie True si la valeur de self est inferieure à celle de carte,
      False sinon    :carte: Objet de type Carte '''
```

b) Travail

Compléter et implémenter la classe carte.
Tester le comportement de la classe Carte.
Afin de tester les méthodes vous implémenterez au choix :

- des assertions,
- une fonction dédiée testCarte() :

voire utiliser un if dédié à l'appel exclusif du script if __name__ == "__main__" :

4) La classe JeuDeCarte (fichier jeu.py).

a) La classe JeuDeCarte

```
from carte import *  # Il faut importer la classe Carte et les variables globales
import random  # Nécessaire pour mélanger le jeu
class JeuCartes():
      def __init__(self, nbCartes=52):
      # Le jeu doit comporter 32 ou 52 cartes, effectuer un contrôle
            self.jeu= [] #self.jeu est une liste de self.nbCartes
            … # à compléter en exécutant la méthode creerJeu() qui doit compléter la
liste
      ### Définition des méthodes d'instances ###
      def getTailleJeu(self):
      ''' Fonction qui retourne le nombre de cartes du jeu Valeur retournée: type int
'''
      def creerJeu(self): # utilise des objet
      '''Créée la liste des cartes de l'attribut self.jeu '''
      def getJeu(self):
      '''Renvoie la liste des cartes correspondant à l'attribut self.jeu'''
      def melanger(self): # utiliser le module random ...
```

```
        '''Mélange sur place les cartes de la liste des cartes associée au champ
self.jeu'''
        def distribuerCarte(self):
        ''' Cette fonction permet de distribuer une carte à un joueur. Elle décrémente le
nb de cartes      du jeu. Valeur retournée: Objet de type Carte '''
        def distribuerJeu(self, nbJoueurs, nbCartes):
        ''' Cette méthode distribue nbCartes à chacun des nbJoueurs'''
        Valeur retournée: une liste de listes de cartes (une liste par joueur)
        # idée : construire la liste des listes des joueurs avec None
```

b) Travail

Compléter et implémenter la classe JeuDeCarte.
Tester le comportement de la classe JeuDeCarte.

5) La classe Joueur (fichier joueur.py).

a) Construire la classe Joueur avec les éléments suivants

La classe Joueur a les attributs suivants :

nom : Nom du joueur
nbCartes : Correspond au nombre de cartes dans la main du joueur
mainJoueur : Liste des cartes (objets de type Carte) dans la main du joueur

Cette classe contient les méthodes suivantes :

getNom() : Accesseur de l'attribut nom
getNbCartes() : Accesseur de l'attribut nbCartes
jouerCarte() : Enlève et renvoie la dernière carte (objet de type Carte) de la main du joueur pour la jouer, ou retourne None s'il n'y a plus de cartes dans la main du joueur
inserer Main() : Fonction qui insère les cartes de la liste des cartes gagnées (le tas) dans la main du joueur. (Remarque, il est possible qu'il y ait un None dans le tas)

b) Travail de vérification

Construire la fonction test qui vérifie le comportement de la classe.
idée: Les tests peuvent se faire avec un objet jeu de cartes mais également avec des listes d'entiers.

6) La classe Bataille (fichier bataille.py).

La classe bataille doit instancier

- un jeu de cartes,
- deux joueurs et leurs cartes (leur main)
- le tas sur la table
- implémenter les méthodes jouer
- et les méthodes d'affichage

On passe également en attribut le nombre de cartes à distribuer à chaque joueur

Tester votre classe :

- instanciation de la classe Bataille
- gestion d'un tour.
- affichage du tour
- gestion du vainqueur