INITIATION A LA PROGRAMMATION OBJET

1. Introduction

Depuis la première, voici un résumé de ce que nous avons appris à faire pour programmer:

- Au début, nous avons d'abord utilisé de simples instructions(`if` , `for` , `while` ,...). Nous avons en quelque sorte « programmé à la main » (c'est-à-dire pratiquement sans outils).
- Nous avons ensuite découvert les fonctions prédéfinies(`print()` , `input()` ,...) et appris qu'il existait ainsi de vastes collections d'outils spécialisés, réalisés par d'autres programmeurs.
- En apprenant à écrire nos propres fonctions(`def` , `return`), nous sommes devenus capables de créer nous-même de nouveaux outils, ce qui nous a donné un surcroît de puissance considérable.

Voici l'étape suivante :

- Nous allons apprendre les bases et le vocabulaire de la programmation objet avec Python, mais la plupart des langages modernes sont des langages objets. C'est comme si nous devenions capables de construire des machines qui produisent des outils.

Qu'est-ce qu'un objet ? :

- Un objet est une entité que l'on construit à partir d'une classe (c'est-à-dire en quelque sorte une « catégorie » ou un « type » d'objet)
- Comme les objets de la vie courante, les objets informatiques peuvent être très simples ou très compliqués. Ils peuvent être composés de différentes parties, qui soient elles-mêmes des objets, ceux-ci étant faits à leur tour d'autres objets plus simples, etc.

Un exemple

In [1]:
```
from turtle import * #1.

tortue=Turtle(shape='turtle') #2.

for i in range(3):
    tortue.forward(100) #3.
    tortue.left(120) #4.

exitonclick()
```

1. On utilise le module `turtle` qui est un fichier qui contient des définitions de classes d'objets(des "modèles" d'objets).
2. On crée une instance de la classe `Turtle()` (un objet appelé `tortue`) et on initialise l'attribut `shape` à la valeur `turtle` .
3. On applique la méthode `forward` avec l'argument `100` à l'objet `tortue` .
4. On applique la méthode `left` avec l'argument `120` à l'objet `tortue` .

A retenir

L'idée de base de la programmation orientée objet consiste à regrouper dans un même ensemble (l'objet), à la fois un certain nombre de données (ce sont les attributs d'instance), et les algorithmes destinés à effectuer divers traitements sur ces données (ce sont les méthodes, à savoir des fonctions particulières encapsulées dans l'objet).

- En résumé : *Objet = [attributs + méthodes]*

2. Classes et attributs

Les classes sont les principaux outils de la programmation orientée objet (Object Oriented Programming ou OOP). Ce type de programmation permet de structurer les logiciels complexes en les organisant comme des ensembles d'objets qui interagissent, entre eux et avec le monde extérieur.

Définition d'une classe

Pour créer une nouvelle classe d'objets Python, on utilise l'instruction `class` . Nous allons donc apprendre à utiliser cette instruction, en commençant par définir un objet : la voiture.

In [3]:
```
class Voiture:
    '''Definition d'une voiture'''
```

Remarques :

- L'instruction `class` nécéssite le double point à la fin de la ligne comme les autres instructions compposées sous python (`if` , `def` , ...)
- Le bloc doit contenir au moins une ligne. Dans cet exemple, il n'y a rien d'autre qu'une description entre les symboles `'''` (docstring).
- Par convention, le nom d'une classe commence par une majuscule

Création d'un objet

Nous pouvons désormais nous servir de cette classe pour créer un objet de cette classe. Voyons ce que nous pouvons en faire :

In [7]:
```
my_car=Voiture()
print(my_car)
print(my_car.__doc__)
```

```
<__main__.Voiture object at 0x00000249176D7D60>
Definition d'une voiture
```

Remarques :

- Après cette instruction, la variable `my_car` contient la référence d'un nouvel objet `Voiture()` . On dit que `my_car` est une instance de la classe `Voiture()` .
- Comme pour les fonctions, les classes auxquelles on fait appel doivent être accompagnées de parenthèses.
- Le premier message indique bien que `my_car` est une instance de la classe `Voiture()` , située à l'adresse mémoire indiquée ici en notation héxadécimale.
- Le deuxième message affiche la documentation de la classe que nous avons nous-mêmes saisi.Nous rencontrerons de nouveau les caractères `__` dans la suite de la feuille

Manipuler les attributs

L'objet que nous venons de créer est juste une coquille vide. Nous allons à présent lui ajouter des composants, à l'aide du symbole `=` et du point `.`

In [8]:
```
my_car.marque='Peugeot'
my_car.modele='504'
my_car.couleur='beige'
my_car.km=200000
my_car.annee=1983
```

In [9]:
```
print(my_car.modele)
print(my_car.annee >2000)
```

```
504
False
```

Remarques:

- Les variables définies ainsi sont directement liées à `my_car` , ce sont les attributs de cet objet. On les appelle également des variables d'instance.
- On peut ensuite utiliser ces attributs comme n'importe quelle expression et les manipuler comme des variables ordinaires.
- ATTENTION : les exemples données ici sont provisoires, ils sont fait pour simplifier cette première approche. La bonne manière de les utiliser est abordée un peu plus loin.

Similitude et unicité

Exercice 1 :

1. Créer une instance `your_car` de la classe `Voiture()` avec les valeurs de variables d'instance suivantes :

- 'Renault', '4L', 'bleu', 150000, 1970

In [12]:
```
#1.
your_car=Voiture()



#
```

1. Créer une instance `her_car` de la classe `Voiture()` avec les mêmes attributs et valeurs que `my_car` (il y a 2 façons de faire). `my_car` et `her_car` sont-ils le même objet ?

In [71]:
```
#2. méthode 1
her_car=Voiture()




print(my_car==her_car,my_car, her_car)

#2. méthode 2




print(my_car==her_car,my_car, her_car)
```

```
False <__main__.Voiture object at 0x000002D299E42490> <__main__.Voiture object at 0x000002D299E4A0A0>
True <__main__.Voiture object at 0x000002D299E42490> <__main__.Voiture object at 0x000002D299E42490>
```

Remarques :

- Dans la langue parlée, le mot « même », a des significations différentes dans les phrases : « Charles et moi avons la même voiture » et « Charles et moi avons la même mère ». Dans la première, ce que je veux dire est que la voiture de Charles et la mienne sont du même modèle. Il s'agit pourtant de deux voitures distinctes. Dans la seconde, j'indique que la mère de Charles et la mienne constituent en fait une seule et unique personne.
- Lorsque nous traitons d'objets logiciels, nous pouvons rencontrer la même ambiguïté. Par exemple, si nous parlons de l'égalité de deux objets `Voiture()` , cela signifie-t-il que ces deux objets contiennent les mêmes données (leurs attributs), ou bien cela signifie-t-il que nous parlons de deux références à un même et unique objet ?
- Dans la méthode 1 de la deuxième question, les instructions créent deux objets dstincts qui ont les mêmes attributs et valeurs (`my_car` et `her_car`) ne désignent pas le même objet, chacun a son adresse mémoire.
- Dans la méthode 2 de la deuxième question, Par l'instruction `her_car=my_car` , nous assignons le contenu de `my_car` à `her_car` . Cela signifie que désormais ces deux variables référencent le même objet. Ces variables sont des alias l'une de l'autre.Lorsque l'on modifie un attribut d'une variable , l'autre est également modifiée.

3. Méthodes

- Comme déjà signalé plus haut, utiliser de cette façon les attributs d'une instance n'est pas une pratique recommandable, car les attributs que nous déclarons à nos objets doivent être valable pour tous les objets de la même classe.
- Nous allons maintenant étudier comment faire fonctionner les objets à l'aide d'outils vraiment appropriés, que nous appellerons des méthodes(qui ressemblent beaucoup aux fonctions).
- L'ensemble de ces méthodes constituera ce que nous appellerons désormais l'interface de l'objet.

Reprenons une nouvelle voiture :

In [10]:
```
new_car=Voiture()
new_car.marque='Ford'
new_car.modele='Ranger'
new_car.couleur='noir'
new_car.km=300000
new_car.annee=2007
```

Exercice 2:

Compléter la fonction `rouler(car,vitesse,temps)` ci-dessous :

In [13]:
```
def rouler(car, vitesse, temps):
    '''Fait rouler une voiture à une vitesse en km/h et pendant un temps en h
    et modifie le kilométrage.
    Parametres:
    car : instance de la classe Voiture
    vitesse : vitesse en km/h, type entier
    temps : temps, en heures, type entier

    Sortie :affiche le nouveau kilométrage
    '''




rouler(new_car,130,2)
rouler(my_car,60,3)
rouler(your_car,240,4)
```

Remarques:

- Cette fonction peut être utilisée pour faire rouler toutes les voitures, c'est à dire toutes les instances de la classe `Voiture()` .
- Il serait donc judicieux d'arriver à encapsuler cette fonction dans la classe `Voiture()` elle-même, de manière à s'assurer qu'elle soit toujours automatiquement disponible, chaque fois que l'on aura à manipuler des objets de cette classe.
- Une fonction que l'on aura ainsi encapsulée dans une classe s'appelle une méthode.

Définition d'une méthode

Ecrivons la méthode `rouler` dans la classe `Voiture` :

In [96]:
```
class Voiture:
    '''Definition d'une voiture'''

    def rouler(self, vitesse, temps):
        '''Fait rouler une voiture à une vitesse en km/h et pendant un temps en h
        et modifie le kilométrage.

        Parametres:
        car : instance de la classe Voiture
        vitesse : vitesse en km/h, type entier
        temps : temps, en heures, type entier

        Sortie :affiche le nouveau kilométrage
        '''

        self.km=self.km+vitesse*temps
        print(self.km)
```

- On définit une méthode comme on définit une fonction, c'est-à-dire en écrivant un bloc d'instructions à la suite du mot réservé `def` , mais cependant avec deux différences :
  - La définition d'une méthode est toujours placée à l'intérieur de la définition d'une classe, de manière à ce que la relation qui lie la méthode à la classe soit clairement établie.
  - La définition d'une méthode doit toujours comporter au moins un paramètre, lequel doit être une référence d'instance, et ce paramètre particulier doit toujours être listé en premier.
- On en principe utiliser un nom de variable quelconque pour ce premier paramètre, mais il est vivement conseillé de respecter la convention qui consiste à toujours lui donner le nom : `self` .Ce paramètre `self` est nécessaire, parce qu'il faut pouvoir désigner l'instance à laquelle la méthode sera associée, dans les instructions faisant partie de sa définition.
- On remarque que la définition d'une méthode comporte toujours au moins un paramètre : self, alors que la définition d'une fonction peut n'en comporter aucun.

Utilisation d'une méthode

Exercice 3 :

Essayons maintenant notre méthode.

1. Exécuter le code suivant.

In [97]:
```
#1.
another_car=Voiture()
another_car.rouler(80,1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-97-3cfc6879e8cb> in <module>
      1 #1.
      2 another_car=Voiture()
----> 3 another_car.rouler(80,1)

<ipython-input-96-2ade75c5a3d7> in rouler(self, vitesse, temps)
     14         '''
     15
---> 16         self.km=self.km+vitesse*temps
     17         print(self.km)

AttributeError: 'Voiture' object has no attribute 'km'
```

1. Modifier ce code pour résoudre l'erreur.

In [98]:
```
#2.



#
```

```
180
```

Remarques :

- À plusieurs reprises, nous avons déjà signalé qu'il n'est pas recommandable de créer ainsi des attributs d'instance par assignation directe en dehors de l'objet lui-même, cette manière est source d'erreurs.
- Nous allons nous arranger pour la méthode `rouler` puisse toujours afficher quelque chose, sans qu'il ne soit nécessaire d'effectuer au préalable une manipulation sur l'objet nouvellement créé.
- En d'autres termes, il faut que les variables d'instance soient prédéfinies elles aussi à l'intérieur de la classe, avec pour chacune d'elles une valeur « par défaut ».
- Pour obtenir cela, nous allons faire appel à une méthode particulière, que l'on désignera par la suite sous le nom de constructeur. Une méthode constructeur est exécutée automatiquement lorsque l'on instancie un nouvel objet à partir de la classe. On peut donc y placer tout ce qui semble nécessaire pour initialiser automatiquement l'objet que l'on crée.

La méthode constructeur

Insérons les caractéristiques d'une voiture dans la classe `Voiture()` :

In [99]:
```
class Voiture:
    '''Definition d'une voiture'''

    def __init__(self,marque='Trabant',modele='601',couleur='gris',km=10000,annee=1961):
        self.marque=marque
        self.modele=modele
        self.couleur=couleur
        self.km=km
        self.annee=annee

    def rouler(self, vitesse, temps):
        '''Fait rouler une voiture à une vitesse en km/h et pendant un temps en h
        et modifie le kilométrage.

        Parametres:
        car : instance de la classe Voiture
        vitesse : vitesse en km/h, type entier
        temps : temps, en heures, type entier

        Sortie :affiche le nouveau kilométrage
        '''

        self.km=self.km+vitesse*temps



a_car=Voiture()
print(a_car.marque)
print(a_car.annee)

the_car=Voiture('Peugeot','e-legend','turquoise',0,2018)
print(the_car.modele)
the_car.rouler(100,10)
print(the_car.km)
```

```
Trabant
1961
e-legend
1000
```

Remarques :

- Afin qu'elle soit reconnue comme telle par Python, la méthode constructeur devra obligatoirement s'appeler `__init__` (deux caractères « souligné », le mot init, puis encore deux caractères « souligné »).
- Cette méthode `__init__()` comporte à présent 6 paramètres, avec pour 5 d'entre eux une valeur par défaut. Nous obtenons ainsi une classe encore plus perfectionnée.
- Lorsque nous instancions un objet de cette classe, nous pouvons maintenant initialiser ses principaux attributs à l'aide d'arguments, au sein même de l'instruction d'instanciation. Et si nous omettons tout ou partie d'entre eux, les attributs reçoivent de toute manière des valeurs par défaut.
- Lorsque l'on écrit l'instruction d'instanciation d'un nouvel objet, et que l'on veut transmettre des arguments à sa méthode constructeur, il suffit de placer ceux-ci dans les parenthèses qui accompagnent le nom de la classe. On procède donc exactement de la même manière que lorsque l'on invoque une fonction quelconque.

Exercice 4 :

1. Ajouter dans le programme ci-dessous la méthode `repeindre` . Cette méthode doit permettre de modifier la couleur d'une voiture.
2. Ajouter la variable d'instance `puissance` avec comme valeur par défaut `26` .
3. Ajouter la méthode `pimp` qui doit permettre d'augmenter la puissance d'une voiture d'un nombre de chevaux souhaité.

In [14]:
```
class Voiture:
    '''Definition d'une voiture'''

    def __init__(self,marque='Trabant',modele='601',couleur='gris',km=10000,annee=1961,puissance=26):
        self.marque=marque
        self.modele=modele
        self.couleur=couleur
        self.km=km
        self.annee=annee
        self.puissance=puissance

    def rouler(self, vitesse, temps):
        '''Fait rouler une voiture à une vitesse en km/h et pendant un temps en h
        et modifie le kilométrage.

        Parametres:
        car : instance de la classe Voiture
        vitesse : vitesse en km/h, type entier
        temps : temps, en heures, type entier

        Sortie :affiche le nouveau kilométrage
        '''

        self.km=self.km+vitesse*temps


    def repeindre(self,nouv_couleur):
        '''Repeind la voiture d'une couleur souhaitée
        parametre :
        nouv_couleur la nouvelle couleur type string
        '''









        pass

    def pimp(self,cv):
        '''Modifie la puissance de la voiture
        parametre : cv , nombre de chevaux en plus, type entier'''




        pass


trabi=Voiture()
trabi.repeindre('rose avec des fleurs')
print(trabi.couleur)
trabi.pimp(120)
print(trabi.puissance)
```

```
gris
26
```

La méthode `__str__`

- Lorsque l'on essaie d'afficher une instance, nous quelque chose qui n'est pas très fonctionnel:

In [103]:
```
car=Voiture()
print(car)
```

```
<__main__.Voiture object at 0x000002D2982A9040>
```

- Pouvoir afficher quelque chose de plus lisible pour un humain, nous allons redédinir le comportement de la fonction `print()` avec les instance de notre classe `Voiture()` grâce à une autre méthode spéciale : la méthode `__str` :

In [105]:
```
class Voiture:
    '''Definition d'une voiture'''

    def __init__(self,marque='Trabant',modele='601',couleur='gris',km=10000,annee=1961,puissance=26):
        self.marque=marque
        self.modele=modele
        self.couleur=couleur
        self.km=km
        self.annee=annee
        self.puissance=puissance

    def __str__(self):
        return self.marque

car=Voiture()
print(car)
```

```
Trabant
```

Remarques :

- Cette méthode coporte un unique parametre `self` , qui désigne l'instance concernée par l'appel de cette méthode.
- On peut y placer les informations de notre choix.
- ce qui est renvoyé est nécéssairement une chaînde de caractères.

Exercice 5:

Modifier le programme ci-dessous pour que la méthode `__str__` renvoie toutes les caractéristiques de la voiture de façon lisible.

In [16]:
```
class Voiture:
    '''Definition d'une voiture'''

    def __init__(self,marque='Trabant',modele='601',couleur='gris',km=10000,annee=1961,puissance=26):
        self.marque=marque
        self.modele=modele
        self.couleur=couleur
        self.km=km
        self.annee=annee
        self.puissance=puissance

    def __str__(self):

        return ''

car=Voiture()
print(car)
```

Autres méthodes spéciales `__`

D'autres méthodes particulières de python mermettent de faciliter la syntaxe, en voici quelques unes. On se réfèrera aux exercices 7 et 9 pour leur utilisation :

méthode	appel	effet
`__lt__`	a < b	renvoie `True` si a est plus petit que b
`__eq__`	a == b	renvoie `True` si a est égal à b
`__add__`	a + b	redéfinit l'opérateur `+`
`__mul__`	a * b	redéfinit l'opérateur `*`

4. A retenir

- La programmation objet structure les programmes en regroupant dans une même entité des données et le code manipulant ces données.
- Dans ce paradigme de programmation, on manipule des objets pouvant contenir plusieurs données sous le forme d'attributs, à l'aide de fonctions particulières appelées méthodes.
- Chaque objet est une instance d'une classe, la classe définissant l'ensemble des attributs et méthodes que possèdent ses instances.

5. Exercices

Exercice 6 : Compte bancaire

Définir une classe `CompteBancaire()` , qui permette d'instancier des objets tels que `compte1` , `compte2` , etc.

1. Le constructeur de cette classe initialisera deux attributs d'instance `nom` et `solde` , avec les valeurs par défaut `'Dupont'` et `1000` .Trois autres méthodes seront définies :
2. `__str__` doit renvoyer le nom du titulaire et le solde de son compte.
3. `depot(somme)` permettra d'ajouter une certaine somme au solde.
4. `retrait(somme)` permettra de retirer une certaine somme du solde.

On complètera le programme ci-dessous au fur et à mesure que l'on pourra tester sur les exemples proposés plus bas.

In [151]:
```
class CompteBancaire:









compte1=CompteBancaire('Prunelle',800)
print(compte1)
compte1.depot(350)
compte1.retrait(20)
print(compte1)

compte2=CompteBancaire()
print(compte2)
compte2.depot(100)
compte2.retrait(150)
print(compte2)
```

```
Prunelle 800
Prunelle 1130
Dupont 1000
Dupont 950
```

Exercice 7: Date

Définir une classe `Date` pour représenter une date:

1. La méthode constructeur comportera trois attributs entiers strictement positifs :
   - `jour` avec la valeur par défaut `1` .
   - `mois` avec la valeur par défaut `1` .
   - `annee` avec la valeur par défaut `1970` .
2. La méthode `__str__` qui renvoie une date du type `'1/1/1970'`
3. La méthode `lt` ("lower than") qui reçoit une deuxième date `d` en argument et qui renvoie `True` si la première date est antérieure à la date `d` .
4. Modifier le nom de la méthode `lt` en `__lt__` , puis utiliser l'opérateur `<` pour comparer deux dates. Epatant non ?

In [213]:
```
class Date:




d1=Date()
d2=Date(20,7,1969)
print(d1,d2)
print(d1<d2, d2<d1, d1<d1)
```

```
1/1/1970 20/7/1969
False True False
```

Exercice 8 : Planètes

Définir une classe `Sphere` . Les objets construits à partir de cette classe seront des Sphères de tailles et de matières variées pouvant représenter des planètes.

1. La méthode constructeur comportera trois attributs :
   - `nom` avec la valeur `'inconnue'` par défaut
   - `rayon` avec la valeur par défaut 1 (exprimée en km)
   - `densité` avec la valeur par défaut 1 (exprimée en g/cm^3)
2. La méthode `__str__()` doit renvoyer le nom de la sphere ainsi que son rayon et sa densité, avec les unités.
3. La méthode `circonference()` doit renvoyer la circonférence de la sphère (en km)
4. La méthode `volume()` doit renvoyer le volume de la sphère (en km^3.
5. La méthode `surface()` doit renvoyer la surface de la sphère (en km^2).
6. La méthode `masse()` doit renvoyer la masse de la sphère (exprimée en tonnes).

In [193]:
```
from math import *

class Sphere:









planete3=Sphere('Terre',6400,5510)
print(planete3)
print('circonference :', planete3.circonference(), ' km')
print('volume :', planete3.volume(), 'km3')
print('surface :', planete3.surface(), 'km2')
print('masse :', planete3.masse(),'tonnes')
```

```
Terre : 6400 km, 5510 kg/m3
circonference : 40212.385965949354  km
volume : 1098066219443.5236 km3
surface : 514718540.3641517 km2
masse : 6.050344869133815e+21 tonnes
```

Exercice 9 : Fraction

Définir une classe `Fraction()` pour représenter un nombre rationnel.

1. La méthode constructeur comportera deux attributs strictement positifs :
   - `numerateur` avec la valeur `2` par défaut.
   - `denominateur` avec la valeur `3` par défaut.
2. La méthode `__str__()` doit renvoyer une écriture du type `'2/3'` .
3. La méthode `eq()` reçoit une deuxième fraction `f` en argument et qui doit renvoyer `True` si la première fraction est égale à la seconde fraction `f` .
4. La méthode `lt()` ("lower than") reçoit une deuxième fraction `f` en argument et qui doit renvoyer `True` si la première fraction est strictement inférieure à la seconde fraction `f` .
5. La méthode `add()` reçoit une deuxième fraction `f` en argument et doit renvoyer la somme de la première et de la deuxième fraction.
6. La méthode `mul()` reçoit une deuxième fraction `f` en argument et doit renvoyer le produit de la première et de la deuxième fraction.
7. Modifier les noms des méthodes des questions 3 à 6 en ajoutant de part et d'autres les symboles `__` .Trouver les opérateurs qui permettent alors de tester ici l'égalité, la comparaison, l'addition et la multiplication de fractions.

On complétera le programme ci-dessous au fur et à mesure et on pourra utiliser les instructions écrites plus bas.

In [215]:
```
class Fraction:
    '''Definit une classe permettant de manipuler des fractions rationnelles
    dont le numerateur et le denominateur sont strictement positifs'''
```

```
#1.
f1=Fraction(2,3)
f2=Fraction(4,6)
f3=Fraction(3,4)

#2.
print(f1,f2,f3)

#3.
#print(f1.eq(f2),f1.eq(f3))
print(f1==f2 , f1==f3)

#4.
#print(f1.lt(f2), f1.lt(f3))
print(f1<f2 , f1<f3)

#5.
#print(f1.add(f2),f1.add(f3))
print(f1+f2 , f1+f3)

#6.
#print(f1.mul(f2),f1.mul(f3))
print(f1*f2 ,f1*f3)
```

```
2/3 4/6 3/4
True False
False True
24/18 17/12
8/18 6/12
```

Exercice 10 : Jeu de cartes 1

Définir une classe `JeuDeCartes()` permettant d'instancier des objets dont le comportement soit similaire à celui d'un vrai jeu de cartes.La classe devra comporter au moins les quatre méthodes suivantes :

1. Méthode constructeur : création et remplissage d'une liste de 52 éléments, qui sont eux-mêmes des tuples de 2 entiers. Cette liste de tuples contiendra les caractéristiques de chacune des 52 cartes. Pour chacune d'elles, il faut en effet mémoriser séparément un entier indiquant la valeur (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, les 4 dernières valeurs étant celles des valet, dame, roi et as), et un autre entier indiquant la couleur de la carte (c'est-à-dire 3,2,1,0 pour Cœur, Carreau, Trèfle et Pique).Dans une telle liste, l'élément $(11,2)$ désigne donc le valet de Trèfle, et la liste terminée doit être du type : `[(2, 0), (3,0), (3,0), (4,0), ..... (12,3), (13,3), (14,3)]`
2. Méthode `nom_carte()` : cette méthode doit renvoyer, sous la forme d'une chaîne, l'identité d'une carte quelconque dont on lui a fourni le tuple descripteur en argument. Par exemple, l'instruction : `print(jeu.nom_carte((14, 3)))` doit provoquer l'affichage de : `As de pique` .
3. méthode `battre()` : comme chacun sait, battre les cartes consiste à les mélanger.Cette méthode sert donc à mélanger les éléments de la liste contenant les cartes, quel qu'en soit le nombre. *Aide : on pourra utiliser la méthode `shuffle` du module random pour les listes*
4. méthode `tirer()` : lorsque cette méthode est invoquée, une carte est retirée du jeu. Le tuple contenant sa valeur et sa couleur est renvoyé au programme appelant. On retire toujours la première carte de la liste. Si cette méthode est invoquée alors qu'il ne reste plus aucune carte dans la liste, il faut alors renvoyer l'objet spécial `None` au programme appelant. *Aide : on pourra utiliser la fonction `del` pour les listes.*

On complètera le programme ci-dessous au fur et à mesure. L'objectif est d'éxécuter le programme écrit sous la définition de la classe sans erreur.

In [147]:
```
from random import *

class JeuDeCartes:
    '''classe définissant un jeu de 52 cartes'''









jeu=JeuDeCartes()
jeu.battre()

for n in range(53):
    carte=jeu.tirer()
    if carte==None:
        print('Terminé')
    else:
        print(jeu.nom_carte(carte), end=' | ')
```

```
Valet de Pique | 10 de Coeur | Dame de Pique | 2 de Carreau | 6 de Pique | 2 de Coeur | Dame de Coeur | 9 de Pique
| Valet de Coeur | 4 de Pique | 3 de Pique | 6 de Carreau | 7 de Coeur | 7 de Carreau | 8 de Pique | 9 de Carreau
| 10 de Pique | Roi de Pique | 6 de Trèfle | Valet de Trèfle | 3 de Trèfle | 8 de Carreau | 10 de Carreau | 10 de
Trèfle | 5 de Coeur | 2 de Trèfle | 9 de Coeur | As de Coeur | 8 de Trèfle | Valet de Carreau | Dame de Trèfle | 5
de Pique | 9 de Trèfle | Roi de Carreau | Roi de Coeur | 5 de Carreau | As de Pique | As de Carreau | 6 de Coeur |
7 de Trèfle | 8 de Coeur | 7 de Pique | 4 de Carreau | 3 de Coeur | Dame de Carreau | As de Trèfle | 4 de Coeur |
5 de Trèfle | 2 de Pique | 4 de Trèfle | 3 de Carreau | Roi de Trèfle | Terminé
```

Exercice 11 : Jeu de cartes 2

A l'aide de la classe de l'exercice précédent:

1. Définir deux joueurs A et B en instanciant deux jeux de cartes et les battre.
2. Ensuite , à l'aide d'une boucle, tirer 52 fois une carte pour chacun des deux joueurs et comparer les valeurs. Si c'est la première des deux qui a la valeur la plus élevée, on ajoute un point au joueur A. Si la situation contraire se présente, on ajoute un point au joueur B. Si les deux valeurs sont égales, on passe au tirage suivant.
3. Au terme de la boucle, comparer les comptes de A et B pour déterminer le gagnant.

In [206]:
```
A=JeuDeCartes()
B=JeuDeCartes()
A.battre()
B.battre()

scoreA=0
scoreB=0









#
```

```
A gagne avec 28 pts
```