1 TD 6 : Classes, héritage

(correction page ??)

Abordé lors de cette séance	
programmation	Classes, héritage
algorithme	...

La séance précédente a montré comment fonctionnait une classe, comment elle s'écrivait. Cette séance est à propos de l'*héritage* qui est une propriété des langages objets. Elle est utile par exemple lorsqu'on doit écrire plusieurs versions d'un même algorithme et qu'une petite partie seulement change d'une version à l'autre.

Supposons que vous ayez un algorithme constitué de trois fonctions plus une dernière qui appelle les trois autres dans le bon ordre. On désire créer une version pour laquelle une des trois fonctions seulement est modifiée.

```
class Version1 :
    def __init__ (self, p) :
        self.p = p
    def fonction1 (self):
        print("Version1.fonction1", self.p)
    def fonction2 (self):
        print("Version1.fonction2", self.p)
    def fonction3 (self):
        print("Version1.fonction3", self.p)
    def fonction_finale (self):
        self.fonction1()
        self.fonction2()
        self.fonction3()

v = Version1(0)
v.fonction_finale()

# affiche
# Version1.fonction1 0
# Version1.fonction2 0
# Version1.fonction3 0
```

On souhaite changer la fonction `fonction2` sans modifier la classe `Version1` et en écrivant le moins possible de code.

```
class Version2(Version1):
    def fonction2 (self):
        print("Version2.fonction2", self.p+1)

v = Version2(0)
v.fonction_finale()

# affiche
# Version1.fonction1 0
# Version2.fonction2 1
# Version1.fonction3 0
```

Le langage a compris qu'on avait changé une fonction et il s'en sert dans la seconde classe. Pour que

cela fonctionne, il faut néanmoins respecter une contrainte essentielle : la fonction remplacée (ou surchargée) doit accepter les mêmes paramètres et retourner le même type de résultat [1].

Première demi-heure : Classes .

Exercice 1

On créer une classe `Piece` qui contient deux méthodes : une méthode `tirage_aleatoire` et une méthode qui appelle la précédente pour faire une moyenne sur n tirages.

```
import random
class Piece :
    def tirage_aleatoire(self, precedent) :
        return random.randint(0,1)
    def moyenne_tirage(self, n):
        # ....
        return ...

p = Piece()
print (p.moyenne_tirage(100))
```

Exercice 2

Le paramètre `precedent` est inutile dans cette première version mais on suppose maintenant que le joueur qui joue est un tricheur. Lorsqu'il perd, il joue une pièce truquée le coup d'après pour laquelle la probabilité d'avoir 1 est de 0,7. On veut implémenter cela avec une classe `PieceTruquee`.

```
import random
class PieceTruquee(Piece) :
    # ....
```

Pour choisir de faire telle ou telle avec une probabilité de 0,7, on peut écrire :

```
import random
if random.random() <= 0.7 :
    # faire une chose avec la probabilité 0.7
else :
    # faire une autre chose avec la probabilité 0.3
```

Seconde demi-heure : Utiliser des méthodes de la classe mère .

Lorsqu'on change une fonction, on a parfois juste d'un petit changement par rapport à la méthode précédente qu'il faut pouvoir appeler. Si on reprend l'exemple précédent, on modifie la méthode `tirage_aleatoire` pour retourner l'autre valeur :

```
import random
class PieceTruquee(Piece) :
    def tirage_aleatoire(self, precedent) :
        return 1 - Piece.tirage_aleatoire(self, precedent)
```

1. Cette contrainte n'est pas obligatoire en *Python* mais elle l'est dans la plupart des langages. Il est conseillé de la respecter.

Exercice 3

Ecrire une classe `PieceTruqueeMix` qui appelle aléatoiremnt soit `Piece.tirage_aleatoire` soit `PieceTruquee.tirage_aleatoire`.

Autre construction possible avec des fonctions

La création de classe peut sembler fastidieuse. Une autre solution est l'utilisation de fonction comme paramètre d'une autre fonction :

```
def moyenne_tirage(n, fonction):
    tirage = [ ]
    for i in range (n) :
        precedent = tirage[-1] if i > 0 else None
        tirage.append( fonction (precedent) )
    s = sum(tirage)
    return s * 1.0 / len(tirage)

print (moyenne_tirage(100, lambda v : random.randint(0,1) ))

def truquee (precedent) :
    if precedent == None or precedent == 1 :
        return random.randint(0,1)
    else :
        return 1 if random.randint(0,9) >= 3 else 0

print (moyenne_tirage(100, truquee ))
```

Exercice 4

Comment utiliser les fonctions dans le cas de la pièce `PieceTruqueeMix`.

Quatrième demi-heure : Interrogation écrite 45 min .

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

Ceci clôt la première partie du cours.