Séance 4: Feulle d'exercices

Philipp Ahrendt

January 6, 2026

Contents

1 Exercice 1: Fractions

Définir une classe `Fraction`, qui représente des fractions entre entiers a/b, où b $\neq$ 0. Cette classe doit contenir les fonctions suivantes:

Fonction	Description
`__eq__`	a/b,c/d $\rightarrow$ `True` si a/b = c/d, `False` sinon
`__add__`	a/b,c/d $\rightarrow$ a/b + c/d
`__mult__`	a/b,c/d $\rightarrow$ a/b $\times$ c/d
`__str__`	a/b $\rightarrow$ envoyer 'a/b' comme chaîne de caractères
`inv`	a/b $\rightarrow$ b/a (si a $\neq$ 0)
`conv`	a/b $\rightarrow$ représentation décimale

Réponse:

```
class Fraction(object):

    def __init__(self,a,b):
        self.num = a
```

```
        self.den = b

    def __eq__(self,other):
        return self.num*other.den == other.num*self.den

    def __add__(self,other):
        return Fraction(self.num*other.den + other.num*self.den, self.den*other.den)

    def __mult__(self,other):
        return Fraction(self.num*other.num, self.den*other.den)

    def __str__(self):
        return f'{self.num}/{self.den}'

    def inv(self):
        return Fraction(self.den,self.num)

    def conv(self):
        return self.num/self.den
```

2 Exercice 2: Tournoi de pingpong

Dans cet exercice, on va implémenter un tournoi de pingpong en python. Dans un *tournoi*, un certain nombre de *joueurs* jouent des *parties* entre eux pour déterminer un vainqueur, ou un classement final.

Déroulement d'une partie de pingpong

- Une partie a deux participants, un joueur `gauche` et un joueur `droite`, avec un score `score_gauche : score droite`. Au début de la partie, le score est `0 : 0`.
- Pendant la partie, les joueurs effectuent une série d'échanges. A la fin d'un échange, le score du gagnant de l'échange monte de 1.
- La partie se termine lorsqu'un joueur atteint un score de 13. Ce joueur gagne la partie.

Le tournoi se fait en mode "ligue".

Déroulement du tournoi

- Le tournoi est une suite de "journées de championnat". Pendant journée, chaque joueur joue une partie contre un autre joueur. A la fin du tournoi, chaque joueur doit avoir affronté chacun des autres exactement une fois. Un joueur ne peut pas jouer plusieurs parties pendant une même journée.

- A la fin de chaque partie, le vainqueur obtient un "point de victoire", qui est rajouté à son compte. De plus, les deux joueurs obtiennent la "différence de score" de la partie (la différence entre leur propre score et le score de l'autre à la fin de la partie). Les points de victoire et les différences de score sont accumulés pour chaque joueur.

- Le classement final est donné par les points de victoire, et par la différence de score accumulée en cas d'égalité.

Exemple:

Place	Joueur	Points	Différence
1.	`A`	8	12
2.	`B`	4	1
3.	`C`	3	-2

Partie `A` vs `C`: `score_final = 8:13`

Place	Joueur	Points	Différence
1.	`A`	8	7
2.	`C`	4	3
3.	`B`	4	1

L'implémentation python consistera en trois classes: `Joueur`, `Partie` et `Tournoi`.

2.1 Question a: La classe `Joueur`.

La classe `Joueur` a la représentation suivante:

- Un *identifiant*: une chaîne de caractères
  - L'identifiant d'un joueur est unique et permet de le distinguer des autres.
- Un *niveau*: Un paramètre numérique entre 0 et 1

  - Sert à estimer ses chances de gagner les échanges d'une partie. Dans une partie, le joueur avec le niveau le plus élevé gagnera plus d'échanges en moyenne.

- Des *points*: le nombre de points de victoire accumulés.
- Une *différence*: La différence de score accumulée.

```
# Les échanges du joueur ont une "qualité" aléatoire, donnée par la distribution "argus
# Cette distribution diffère selon le paramètre "niveau"
from scipy.stats import argus

class Joueur(object):

    # Ceci est une variable globale partagée par tous le objets de la classe
    # C'est un dictionnaire contenant tous les joueurs créés, indexés par leur identifi
    # Sert à assurer que chaque identifiant n'apparait qu'une seule fois
    ids = {}

    def __init__(self, identifiant, niveau):
        assert identifiant not in Joueur.ids, "joueur déjà existant"

        self.identifiant = identifiant
        self.niveau = niveau
        self.points = 0
        self.diff = 0

        Joueur.ids[identifiant] = self

    # Fonction utilisée pour faire les échanges entre joueurs
    # Pour "jouer", un joueur donne une variable aléatiore suivant la loi "argus", avec
    # Plus le niveau est grand, plus le résultat est grand en moyenne
    def joue(self):
        return argus.rvs(self.niveau)

    # Réponse
    # ------------------------------------------------------------------------------
    def gagne(self, score_perdant):
        self.points += 1
        self.diff += 13 - score_perdant
```

```
def perd(self, score_perdant):
    self.diff -= 13 - score_perdant

def __lt__(self,other):
    if self.points == other.points:
        return self.diff < other.diff
    else:
        return self.points < other.points
# ------------------------------------------------------------------------------

# Pour réinitialiser lors d'un tournoi
# ------------------------------------------------------------------------------
def reinit(self):
    self.points = 0
    self.diff = 0

def __str__(self):
    return self.identifiant
```

Rajouter à la classe `Joueur` ci-dessus les fonctions suivantes:

- `gagne(self, score_perdant)`
  - modifie les attributs du joueur s'il a gagné un match et le score de l'adversaire est `score_perdant`.
- `perd(self, score_perdant)`
  - modifie les attributs du joueur s'il a perdu un match et son propre score est `score_perdant`.
- Une comparaison avec `<` entre deux joueurs.
  - Doit ordonner les joueurs selon les règles de classement du tournoi.

2.2 Question b: La classe `Partie`.

Les attributs de `Partie` sont:

- Deux joueurs, "gauche" et "droite".
- Un score.

- Un booléen `finie`, qui vaut `False` tant que la partie n'est pas finie.
- Un gagnant et un perdant, déterminés à la fin de la partie.

```
class Partie(object):

    def __init__(self, joueur_gauche, joueur_droite):

        self.joueur_gauche = joueur_gauche
        self.joueur_droite = joueur_droite
        self.score = {"gauche":0, "droite":0}
        self.finie = False
        self.gagnant = None
        self.perdant = None

    # Réponse 1
    # ------------------------------------------------------------------------------
    def echange(self):
        gauche = self.joueur_gauche.joue()
        droite = self.joueur_droite.joue()

        if gauche > droite:
            self.score["gauche"] += 1
        else:
            self.score["droite"] += 1

    def fin(self):
        self.finie = True
        if self.score["gauche"] == 13:
            self.gagnant = self.joueur_gauche
            self.perdant = self.joueur_droite
        else:
            self.gagnant = self.joueur_droite
            self.perdant = self.joueur_gauche
    # ------------------------------------------------------------------------------

    def __str__(self):
        return self.joueur_gauche.identifiant + " vs " + self.joueur_droite.identifiant
```

Rajouter à `Partie`

1. Une fonction `echange(self)`

   - contient les instructions à suivre lors d'un échange. Elle doit comparer les variables aléatoires générés par `joue()` pour les deux joueurs et modifier le score selon le résultat.

2. Une fonction `fin(self)`

   - contient les instructions a suivre à la fin de la partie. Elle doit déclarer la partie comme "finie" et déterminer le gagnant et le perdant.

Écrire ensuite une fonction `jouer(partie)`

- Prend en entrée une partie et la joue selon les règles d'une partie. Après la partie elle doit également modifier les points et différences des joueurs selon les règles du tournoi.

Réponse:

```
def jouer(partie):
    while partie.score["gauche"] < 13 and partie.score["droite"] < 13:
        partie.echange()

    partie.fin()

    score_perdant = min(partie.score["gauche"], partie.score["droite"])

    partie.gagnant.gagne(score_perdant)
    partie.perdant.perd(score_perdant)
```

2.3 Question c: création d'un calendrier

On veut créer un tournoi à partir d'une liste de joueurs. Pour cela, il faut créer le calendrier des journées de championnat.

On se limite à des nombres paires de joueurs, supérieures ou égales à 2. Un algorithme pour implémenter cela est illustré par la graphique ci-dessous:

```
Exemple pour 8 joueurs:

1    2    3    4
|    |    |    |    Une journée de tournoi
|    |    |    |
```

```
5    6    7    8

1 <- 2 <- 3 <- 4    Tous les autres on décale d'un pas suivant les flèches (7 va vers 4
|            >
V           /
5 -> 6 -> 7    8 -- On fixe une des places

2    3    4    7
|    |    |    |    Prochaine journée de tournoi
|    |    |    |
1    5    6    8
```

Écrire une fonction `calendrier_parties(joueurs)` qui prend en entrée une liste de joueurs et renvoie un calendrier. Cette fonction doit rendre Une liste de listes, où chaque liste contient les parties (des objets `partie`) à effectuer pendant la journée de tournoi correspondante.

Réponse:

```
def calendrier_parties(joueurs):
    parties_debut = []
    i = 0
    while i < len(joueurs):
        parties_debut.append(Partie(joueurs[i],joueurs[i+1]))
        i += 2
    parties = [parties_debut]
    for k in range(1,len(joueurs)-1):
        parties_suivant = []
        if len(joueurs) == 4:
            parties_suivant.append(Partie(parties[k-1][0].joueur_droite, parties[k-1][
            parties_suivant.append(Partie(parties[k-1][0].joueur_gauche, parties[k-1][
        else:
            parties_suivant.append(Partie(parties[k-1][0].joueur_droite, parties[k-1][
            for l in range(1,len(parties[k-1])-2):
                parties_suivant.append(Partie(parties[k-1][l-1].joueur_gauche, parties
            parties_suivant.append(Partie(parties[k-1][-3].joueur_gauche, parties[k-1]
            parties_suivant.append(Partie(parties[k-1][-2].joueur_gauche, parties[k-1]
        parties.append(parties_suivant)
    return parties
```

2.4 Question d: La classe `Tournoi`.

Initialisation: `Tournoi(participants)`

- le paramètre d'entrée participants est une liste de longueur paire au moins 2 de joueurs.

Les attributs de `Tournoi`:

- Une liste de participants.
- Un calendrier des journées de championnat.
- Un classement
  - Une liste triée dans l'ordre décroissant des participants. L'ordre initial n'est pas important.
- La journée de championnat actuelle.
  - Initialement 0, puis elle augmente de 1 après chaque journée de championnat.
- Un booléen qui indique si le tournoi est fini.

Lors de l'initialisation, les points et différences des joueurs doivent tous être remis à 0.

Méthodes de `Tournoi`:

- `simuler_journee(self)`
  - Tant que le tournoi n'est pas fini, cette fonction effectue les parties de la journée, met à jour le classement et renvoie en sortie les résultats de la journée.

Implémenter la classe `Tournoi` selon les consignes ci-dessus.

Écrire également une fonction `simulation(tournoi)` qui prend en entrée un tournoi, et affiche pour chaque journée les parties et leurs résultats, ainsi que le classement intermédiaire après la journée. A la fin, elle doit afficher le gagnant.

Indication: Pour répondre à cette question, il peut être nécessaire de rajouter des méthodes à la classe

Réponse:

```
# Un exemple de tri pour mettre à jour le classement
def tri_insertion(L):
    for i in range(len(L)):
        k = 0
        ins = L[i]

        while k <= i and L[k] > ins:
            k += 1

        tmp = 0

        while k <= i:
            tmp = L[k]
            L[k]= ins
            ins = tmp

            k += 1

class Tournoi(object):

    def __init__(self, participants):

        # Pour que le tournoi commence à zéro pour tous
        for joueur in participants:
            joueur.reinit()

        self.participants = participants
        self.calendrier = calendrier_parties(participants)
        self.classement = participants
        self.journee = 0
        self.fini = False

    def simuler_journee(self):

        if not self.fini:
            for partie in self.calendrier[self.journee]:
                jouer(partie)

            resultats = self.calendrier[self.journee]
```

```
            tri_insertion(self.classement)
            self.journee += 1

            if self.journee == len(self.calendrier):
                self.fini = True

            return resultats


def simulation(tournoi):
    while not tournoi.fini:
        print("jour", tournoi.journee + 1)
        resultats_journee = tournoi.simuler_journee()

        for partie in resultats_journee:

            score = partie.score

            print(partie, " | ", score["gauche"], ":", score["droite"])

        print("")

        cl = tournoi.classement

        print("classement jour", tournoi.journee)
        for i in range(len(cl)):
            print(str(i+1)+".", cl[i], cl[i].points, cl[i].diff)

        print("")

    classement_final = tournoi.classement
    print(classement_final[0], "a gagné le tournoi")
```

2.5 Pour tester

Voici une liste de 8 joueurs, que vous pouvez utiliser pour tester vos fonctions (les noms viennent du classement mondial du pingpong).

```
niveaux = [1+i/8 for i in range(8)]
```

```
pitchford = Joueur("Pitchford",niveaux[0])
kallberg = Joueur("Kallberg",niveaux[1])
jorgic = Joueur("Jorgic",niveaux[2])
qiu = Joueur("Qiu",niveaux[3])
harimoto = Joueur("Harimoto",niveaux[4])
calderano = Joueur("Calderano",niveaux[5])
lebrun = Joueur("Lebrun",niveaux[6])
wang = Joueur("Wang",niveaux[7])
```