Correction TD Maison

Programmation Orientée Objet

Exercice 1

```
class Date:
    NB_JM=[31,28,31,30,31,30,31,31,30,31,30,31]
    def __init__(self,j,m,a):
        self.jour=j
        self.mois=m
        self.an=a
    def __str__(self):
        F="{}/{}/{}"
        return(F.format(self.jour,self.mois,self.an))
    def Bissextile(self):
        if self.an%100==0:
            return(self.an%400==0)
        else:
            return(self.an%4==0)

     #méthode 1
    def NB_Jours_Ecoules(self):
        nb=self.jour
        if self.mois>1:
            nb+=sum(Date.NB_JM[: self.mois-1])
            if self.Bissextile()and self.mois>2:
                nb+=1
        return(nb)
    # méthode 2
    def NB_Jours_Ecoules(self):
        nb=self.jour
        if self.mois>1:
            for i in range(self.mois-1) :
                nb+=Date.NB_JM[i]
            if self.Bissextile()and self.mois>2:
                nb+=1
        return(nb)

    def NB_Jours_Entre(self,D):
        return(abs(self.NB_Jours_Ecoules()-D.NB_Jours_Ecoules()))

    def __gt__(self,D):
        return(self.NB_Jours_Ecoules()> D.NB_Jours_Ecoules())

class  JoursFeries:
    NB_JM=[31,28,31,30,31,30,31,31,30,31,30,31]
    def __init__(self):
```

```
    self.LD=[]

def Ajout_Date(self):
    #Créer une date valide
    while True:
        try:
            j=int(input('donner le jour'))
            m=int(input('donner le mois'))
            D=Date(j,m,2021)
            print(D)
            if 1<=m<=12 and 1<=j<=JoursFeries.NB_JM[m-1] and
                (self.LD==[] or (self.LD!=[] and D> self.LD[-1])):                    break
        except: continue
    # l'ajouter
    self.LD.append(D)

def __str__(self):
    ch=''
    for D in self.LD:
        ch+=str(D)+' '
    return(ch)
    #ou bien
    L=[str(D) for D in self.L]
    return(' '.join(L))

def EcartMax(self):
    #on calcule le nombre de jours écoulés de chacune des dates de self.LD
    L=[D.NB_Jours_Ecoules() for D in self.LD]
    Em=0
    for i in range(len(L)-1):
        # on compare la valeur de Em et la différence entre les nombres de jours écoulées
         # successifs ( d'indice i et i+1)
        if L[i+1]-L[i]>Em:
            Em=L[i+1]-L[i]
            D1=self.LD[i]
            D2=self.LD[i+1]
    return(Em,D1,D2)

#méthode 2
def EcartMax(self):
    Em=0
    for i in range(len(self.LD)-1):
        # on applique la méthode NB_Jours_Entre de la classe Date
        # sur une 1ère Date: self.LD[i] et une deuxième Date: self.LD[i+1]
        E=self.LD[i].NB_Jours_Entre(self.LD[i+1])
        if E>Em:
```

```
            Em=E
            D1=self.LD[i]
            D2=self.LD[i+1]
        return(Em,D1,D2)

# programme principal
JF=JoursFeries()
while True:
    JF.Ajout_Date()
    # saisir le choix
    while True:
        choix=input('taper C pour continuer et A pour arreter')
        if choix in ['C','A']: break
    if choix=='A': break
print(JF) # appel à __str__ de la classe joursFeries
Emax,D1,D2=JF.EcartMax()
print("l'ecar max est {} entre {} et {}".format( Emax,D1,D2))
```

Exercice 2

```
from random import *
class Pile:
    def __init__(self):
        self.L=[]

    def Empiler(self,e):
        self.L.append(e)

    def Vide(self):
        return(self.L==[])

    def Depiler(self):
        if not self.Vide():  return(self.L.pop())
        else:   return(None)

    def Sommet(self):
         if not self.Vide():  return(self.L[-1])
         else: return(None)

    def Couper(self):
        k=randint(1,len(self.L))
        P1=Pile()
        for i in range(k):
            P1.Empiler(self.Depiler())
        P2=Pile()
        while not self.Vide():
            P2.Empiler(self.Depiler())
```

```
        return(P1,P2)

    def Melanger(self,P):
        P3=Pile()
        while not self.Vide() and not P.Vide():
            k=randint(1,2)
            if k==1: P3.Empiler(self.Depiler())
            else: P3.Empiler(P.Depiler())
        while not self.Vide():
            P3.Empiler(self.Depiler())
        while not P.Vide():
            P3.Empiler( P.Depiler())
        return(P3)
```

```
class Carte:
    def __init__(self,s,n,v):
        self.symbole=s
        self.nature=n
        self.valeur=v

    def __str__(self):
        return(' {} de {}'.format(self.nature,self.symbole))

    def __gt__(self,C):
        return(self.valeur > C.valeur)

    def __add__(self,C):
        return(self.valeur+ C.valeur)
```

```
class JeuCartes:
    L1=['As',2,3,4,5,6,7,8,9,10,'valet','Dame','Roi']
    L2=['carreaux' ,'cœur','pique','trèfle']
    def __init__(self):
        self.P=Pile()
        for s in JeuCartes.L1:
            val=1
            for n in JeuCartes.L2:
                self.P.Empiler( Carte(s,n,val))
                val+=1
        self.P1=Pile()
        self.P2=Pile()

    def initPartie(self):
        for i in range(3):
            P1,P2=self.P.Couper()
            self.P=P1.Melanger(P2)
```

```
    def Demarrer(self):
        while not self.P.Vide():
            self.P1.Empiler( self.P.Depiler())
            self.P2.Empiler( self.P.Depiler())

    def Bataille(self, score1,score2,val):
        print('Début bataille')
        s=2*val
        while not self.P1.Vide(): # ou self.P2.Vide
            # tirer les 2 cartes et les afficher
            C1=self.P1.Depiler()
            print('joueur1:', C1)
            C2=self.P2.Depiler()
            print('joueur 2:',C2)
            # tester les valeurs des cartes tirées
            if C1>C2:  # appel à __gt__
                score1+=s+(C1+C2)   # appel à __add__
                print('fin bataille')
                break
            elif C2>C1:
                score2+=s+(C1+C2)
                print('Fin Bataille')
                break
            else:s+=C1+C2
        return(score1,score2)

    def Jeu(self):
        score1=score2=0
        while not self.P1.Vide():
            # tirer les 2 cartes et les afficher
            C1=self.P1.Depiler()
            print('joueur1:', C1)
            C2=self.P2.Depiler()
            print('joueur 2:',C2)
            # tester les valeurs des cartes tirées
            if C1>C2:  # appel à __gt__
                score1+=C1+C2   # appel à __add__
            elif C2>C1:
                score2+=C1+C2
            else:
                score1,score2=self.Bataille(score1,score2,C1.valeur)
        return(score1,score2)

# Programme principal
J1=JeuCartes()
```

```
J1.initPartie()
J1.Demarrer()
score1,score2=J1.Jeu()
if score1> score2: print('le joueur 1 gagne')
elif score2>score1: print('le joueur 2 gagne')
else: print('égalité')
```

Exercice 3

```
class PolynomeCreux:
    def __init__(self,d=dict()):
        self.data=d
    def Ajout_monome(self):
        #la puissance ne doit pas être parmi les puissances dans le dictionnaire self.data
        while True:
            try:
                p=int(input("donner une puissance"))
                if p>=0 and p not in self.data:
                    break
            except:
                continue
        # le coefficient est un entier strictement positif
        while True :
            try:
                c=int(input("donner un coeff"))
                if type(c)==int:
                    break
            except :
                continue
        # Ajout du nouveau couple au dictionnaire self.data
        self.data[p]=c

    def degree(self):                    #ou bien
        deg=0
        for cle in self.data:
            if cle> deg:
                deg=cle
        return(deg)

    def __call__(self,x):
        s=0
        for puiss,coef in self.data.items():
            s+=coef*x**puiss
        return(s)
```

```
def degree(self):
    L=list(self.data.Keys())
    return(max(L))
```

```
def __add__(self,P):
    # pour créer le polynome somme on doit d'abord créer son attribut D
    #le dictionnaire contenant les différents monômes qui le composent
    D=dict()
    for puiss,coef in self.data.items():
        if puiss in P.data:
            coef2=P.data[puiss]
            if coef+coef2 !=0:
                D[puiss]= coef+coef2
        else:
            D[puiss]= coef

    for puiss,coef in P.data.items():
        if puiss not in self.data:
            print(puiss)
            D[puiss]=coef
            print(D)
    P= PolynomeCreux(D)
    return(P)

def __mul__(self,P):
    D=dict()
    #chaque monôme du premier polynôme doit être multiplié par tous les monômes du deuxième
    for puiss1,coef1 in self.data.items():
        for puiss2,coef2 in P.data.items():
            puiss=puiss1+puiss2
            coef=coef1*coef2
            if puiss not in D and coef!=0:
                D[puiss]=coef
            elif puiss in D and coef!=0:
                D[puiss]+=coef
                if D[puiss]==0:
                    del (D[puiss])
    P=PolynomeCreux(D)
    return(P)

from copy import copy
def __str__(self):
    ch=""
    D=copy(self.data)
    while D!=dict():
        #recherche de la puissance maximale
        pm=-1
        for puiss in D:
            if puiss>pm:
```

```
            pm=puiss
            cm=D[puiss]
        # supprimer ce couple du dictionnaire
        del(D[pm])
        # mettre a jour la chaine
        if pm==0 :
          if cm>0: ch+='+'+str(cm)
          else: ch+=str(cm)
        elif pm==1 :
          if  cm==-1: ch+="-x"
          elif cm==1: ch+="+x"
          elif cm>0:  ch+='+'+str(cm)+'*x'
          elif cm<0:  ch+=str(cm)+'*x'
        elif pm>1:
          if cm==1:ch+='+ x**'+str(pm)
          elif cm==-1:ch+='- x**'+str(pm)
          elif cm>0 : ch+='+'+str(cm)+'*x**'+str(pm)
          else:ch+=str(cm)+'*x**'+str(pm)
        if ch[0]=='+': ch=ch[1:]
    return(ch)
```

Remarque : la méthode __str__

Pour trouver la puissance maximale on peut proposer différentes méthodes :

#méthode 1

```
def __str__(self):
    ch=""
    D=copy(self.data)
    while D!=dict():
        #recherche de la puissance maximale
        pm=-1
        for puiss in D:
            if puiss>pm:
                pm=puiss
                cm=D[puiss]
        # supprimer ce couple du dictionnaire
        del(D[pm])
        # mettre a jour la chaine…
```

#ou bien

```
def __str__(self):
    ch=""
    D=copy(self.data)
    while D!=dict():
        #recherche de la puissance maximale
        L=list(self.data.Keys())

        pm=max(L))

        # supprimer ce couple du dictionnaire
        del(D[pm])
        # mettre a jour la chaine…
```

#méthode 2

```
def __str__(self):
    ch=""
    D=copy(self.data)
    while D!=dict():
        #créer un objet PolynomeCreux avec le nouveau dictionnaire
        P=PolynomeCreux(D)
        #Appliquer la méthode  degree
        pm=P.degree()
```

```
        # supprimer ce couple du dictionnaire
        del(D[pm])
        # mettre a jour la chaine…
#méthode 3
def __str__(self):
    ch=""
    # Extraire tous les clés puis les ordonnés dans l'ordre décroissant
    L=list(self.data.keys())
    L.sort(reverse=True)# Tri dans l'ordre décroissant
    # parcourir les puissances : les clés de self.data et prendre les coefficients associés
    for puiss in L :
      coef=self.data[puiss]
            # mettre a jour la chaine…
```

Exercice 4

```
class Robot:
    def __init__(self,n,x,y,d):
        self.nom=n
        self.x=x
        self.y=y
        self.dir=d

    def Avance(self):
        if self.dir=="Nord":     self.y+=1
        elif self.dir=="Sud":    self.y-=1
        elif self.dir=="Est":     self.x+=1
        else:        self.x-=1

    def Droite(self):
        if self.dir=="Nord":  self.dir="Est"
        elif self.dir=="Est":    self.dir="Sud"
        elif self.dir=="Sud":     self.dir="Oust"
        else:          self.dir="Nord"

    def __str__(self):
        F="le robot {} se trouve dans la pos {},{},  il se dirige vers le {}"
        return(F.format(self.nom,self.x,self.y,  self.dir))

class RobotNG(Robot):
    def __init__(self,n,x,y,d):
        Robot.__init__(self,n,x,y,d)

    def Avance(self,n):
        for i in range(n):
            Robot.Avance(self)

 def Droite(self):
```

```
        Robot.Droite(self)

    def Gauch(self):
        if self.dir=="Nord":       self.dir="Ouest"
        elif self.dir=="Ouest":    self.dir="Sud"
        elif self.dir=="Sud":      self.dir="Est"
        else:     self.dir="Nord"

    def __str__(self):
        return(Robot.__str__(self))

# Programme Principal
L,LR=[],[]
for i in range(20):
    while True:
        try:
            choix=int(input('donner choix'))
            if choix in [0,1]:
                break
        except:
            continue
    # saisir les attributs d'un objet robot ou  robotNG en respectant les conditions de saisie
    nom=input('donner le nom')
    while True:
        try:
            x=int(input("donner l'abscisse"))
            y=int(input("donner l'ordonnée"))
            if type(x)==int and type(y)==int:
                break
        except:
            continue
    while True:
        dir=input("donner la direction")
        if dir in ("Nord","Sud","Est","Ouest"):
            break
    # on crée un objet robot ou robotNG selon  le choix
    if choix==0:   R=Robot(nom,x,y,dir)
    else:    R=RobotNG(nom,x,y,dir)
    L.append(choix)
    LR.append(R)

for i in range(20):
    if L[i]==0: # robot ancienne génération
        print(LR[i].nom,' est un ancien robot')
        print(LR[i])
        # on peut aussi appliquer les méthode  avance et droite
        LR[i].Avance()
        print('nouvelle position')
        print(LR[i])
```

```
else:  # robot nouvelle génération
    print(LR[i].nom,' est un robot nouvelle génération')
    print(LR[i])
    # on peut aussi appliquer les méthodes  avance ,droite et gauche
    LR[i].AvanceNG(6)
    print('nouvelle position')
    print(LR[i])
```