Eléments de correction du DM 1 - T NSI

Ex 1 (6 points)

1. (2) Il y a $\binom{64}{5}$ mains de 5 cartes différentes dans un jeu de 64 cartes.

```
>>> comb(64,5)
7624512
```

   Il y a $\binom{49}{5}$ mains de 5 cartes différentes dans un jeu de 64 cartes.

```
>>> comb(49,5)
1906884
```

2. (a) (1) Il y a $\binom{n}{2}$ couples de sommets dans un polygone à n sommets.

   Or $\binom{n}{2} = \dfrac{n!}{(n-2)!2!} = \dfrac{n(n-1)}{2}$

   (b) Parmi ces $\dfrac{n(n-1)}{2}$ couples, il y a n côtés. Le reste ce sont les diagonales, donc le nombre de diagonales dans un polygone de n cotés est
   $\dfrac{n(n-1)}{2} - n = \dfrac{n(n-1)-2n}{2} = \dfrac{n(n-3)}{2}$

3. (1)

```
def nb_diag(n):
    return n*(n-3)//2
```

4. (2)

```
def nb_diag_rec(n):
    if n == 4:
        return 2
    return nb_diag_rec(n-1) + n-2
```

5. `nb_diag(100)` et `nb_diag_rec(100)` renvoient 4850

Ex 2 (5)

1. (1 + 1) Il y a dans le dessin $2^{11} - 1 = 2047$ carrés
2. 3

```
SEUIL = 10
def arbre(x,y,cote):
    if cote < SEUIL:
        pd()
        carre(cote)
        pu()
    else:
        #dessiner le carre en partant de A
        #elle regarde dans la direction du vecteur
```

```
        pd()
        carre(cote)
        pu()

        #emmener la tortue en E
        #elle regarde dans la direction du vecteur
        left(90)
        forward(cote)
        right(45)
        #mémoriser  l'état de la tortue en E
        x = xcor()
        y = ycor()
        h = heading()
        #appel récursif
        arbre(x,y,cote/sqrt(2))
        #retour : remettre la tortue dans l'état E
        setx(x)
        sety(y)
        setheading(h)
        #aller en F
        forward(cote/sqrt(2))
        right(90)
        x = xcor()
        y = ycor()
        #appel récursif en F
        arbre(x,y,cote/sqrt(2))
```

Ex 3 (5 points)

1.
```
def somme(liste1:list,liste2:list)->list:
    if len(liste1) > len(liste2):
        return [liste1[i] + liste2[i] \
        for i in range(len(liste2))]+ \
        [liste1[i] for i in range(len(liste2),len(liste1))]
    return [liste1[i] + liste[i] \
    for i in range(len(liste1))]+ \
    [liste2[i] for i in range(len(liste1),len(liste2))]

class Polynome:
    def __init__(self,deg,coeff):
        self.deg = deg
        self.coeff = coeff

    def somme(self,other):
        somme_liste = somme(self.coeff, other.coeff)
        i = len(somme_liste) - 1
```

```
        #on enlève les zéros non
        #significatifs
        while i > 0 and somme_liste[i] == 0:
            somme_liste.pop()
            i -= 1
        return Polynome(i,somme_liste)

    def derive(self):
        derive_liste = [0]*(self.deg)
        for i in range(self.deg-1,0,-1):
            derive_liste[i] = self.coeff[i+1]*(i+1)
        return Polynome(self.deg - 1,derive_liste)
```

Ex 4 (4 points)

```
class Cellule_2:
    def __init__(self,p,v,s):
        self.pred = p
        self.val = v
        self.succ = s

def rechercher_2(liste:Cellule_2,val:int) -> Cellule_2:
    adr = liste
    while adr is not None and adr.val != val:
        adr = adr.succ
    return adr

def inserer_2(liste:Cellule_2,val:int)-> Cellule_2:
    if liste is None:
        return Cellule_2(None,val,None)
    liste.pred = Cellule_2(None,val,liste)
    return liste.pred

def supprimer_2(liste:Cellule_2,adr:Cellule_2)-> Cellule_2:
        if adr.pred is not None:
           adr.pred.succ = adr.succ
        else:
            liste = adr.succ
        if adr.succ is not None:
            adr.succ.pred = adr.pred
        return liste
```