

DM 1 - T NSI (à rendre le 04/10/23)

Ex 1

Le but de l'exercice est de calculer de manière **récursive** le nombre de diagonales dans un polygone régulier.

Une diagonale est un segment qui joint deux sommets qui **ne sont pas voisins**, **c'est à dire qui ne sont pas reliés par un côté**.

Dans un carré il y en a 2, et dans un pentagone régulier 5, etc..

1. $\binom{n}{p}$ (lire "*p parmi n*" permet de compter le nombre de parties à p éléments dans un ensemble à n éléments (sans ordre)).

Par exemple il y a $\binom{32}{5}$ mains de 5 cartes différentes dans un jeu de 32 cartes.

Comment calculer $\binom{32}{5}$?

Pour l'instant on utilisera la fonction `comb` du module `math` de Python (pour combinatorics)

<https://docs.python.org/3/library/math.html#math.comb>

```
>>> from math import comb
>>> comb(32,5)
201376
```

Combien de mains de 5 cartes différentes dans un jeu de 64 cartes ?

Combien de combinaisons différentes de 5 numéros dans une grille du Loto à 49 numéros ?

(D'un point de vue mathématique $\binom{n}{p} = \frac{n!}{(n-p)!p!}$. On reviendra plus tard en terme de complexité sur le problème du calcul de $\binom{n}{p}$)

2. Si on numérote les sommets d'un polygone de 1 jusqu'à n , alors tout couple de sommet est **soit un côté, soit une diagonale**

(a) Combien y a-t-il de couples différents en tout dans un polygone de n sommets ?

(b) Combien y a-t-il de côtés dans un polygone de n sommets ?

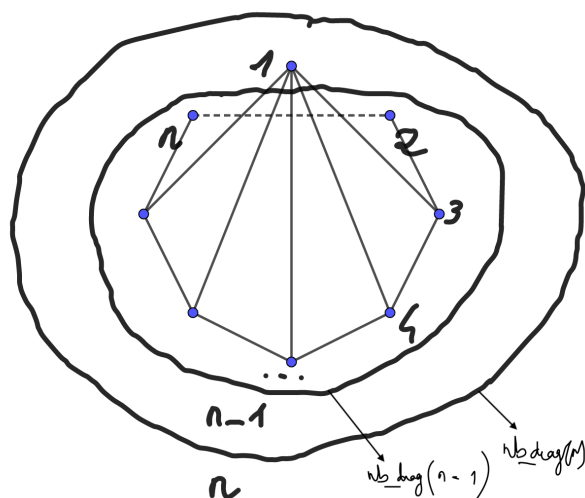
(c) En déduire que le nombre de diagonales dans un polygone de n sommets vaut $\frac{n(n-3)}{2}$. (Vérifier que $\binom{n}{2} = \frac{n(n-1)}{2}$)

Ecrire une fonction `nb_diag(n:int)-> int` qui renvoie $\frac{n(n-3)}{2}$ où n est le nombre de sommets du polygone

3. Ecrire une fonction **récursive** `nb_diag_rec(n:int)->int` en complétant sur votre feuille les pointillés

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

- Ecrire le cas de base (lignes 2 et 3)
- Ecrire l'appel récursif sur le cas précédent (ligne 4 à gauche du signe +)
- Que faut il rajouter à la ligne 4 après le retour de l'appel récursif? (voir dessin ci-dessous)

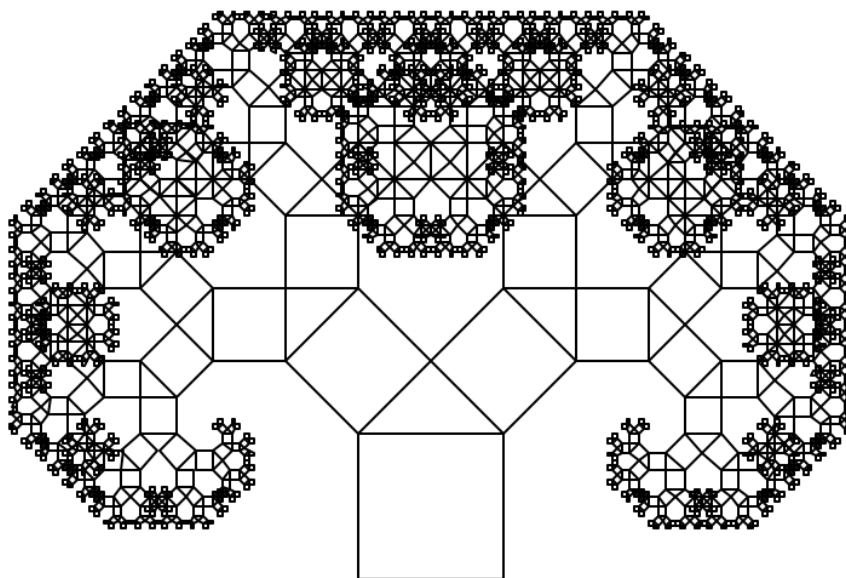


4. Que valent `nb_diag(100)` et `nb_diag_rec(100)` ?

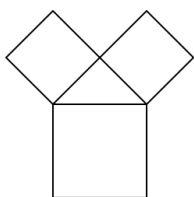
Ex 2

Le but de l'exercice est d'écrire une fonction récursive permettant de réaliser un dessin récursif appelé Arbre de Pythagore.

Ce dessin est réalisé à partir de carrés.



Le motif de base avec des carrés est :



-

- Entre les trois carrés il y a un triangle isocèle rectangle de côté $\frac{cote}{\sqrt{2}}$

The diagram shows a quantum circuit for a 2-qubit system. The input is labeled $A(x, y)$. The circuit consists of a Hadamard gate on the first qubit, followed by a CNOT gate with the first qubit as control and the second qubit as target. The output is labeled E and F .

```

SEUIL = 10
def arbre(x,y,cote):
    # condition d'arrêt
    if cote < SEUIL:
        carre(cote)
    else:
        #dessiner le carre en partant de A
        #elle regarde dans la direction du vecteur
        carre(cote)
        #emmener la tortue en E
        #elle regarde dans la direction du vecteur
        .....
        .....
        .....
        #mémoriser l'état de la tortue en E
        x = xcor()
        y = ycor()
        h = .....
        # appel récursif en E
        arbre(x,y,c/sqrt(2))
        #remettre la tortue dans l'état E
        .....
        .....
        .....
        #emmener la tortue dans l'état F
        ....
        ....
        #appel récursif en F
        x = xcor()
        y = ycor()
        arbre(x,y,c/sqrt(2))

```

Pourquoi il est inutile de mémoriser l'état de la tortue en F ?

Ex 3

Le but de cet exercice est d'écrire une classe `Polynome` pour les entiers.

Un polynôme pour les entiers est une expression de forme générale, où les a_i sont des entiers.

$$\sum_{k=0}^{k=n} a_k x^k = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n.$$

On dit que l'exposant de degré le plus élevé est le **degré du polynôme**

Par exemple $4x^2 - 1$ est un polynôme de degré 2 et $x^5 - 3x^2 + 1$ de degré 5

1. La classe a deux **attributs** : `deg` de type `int` et `coeff` de type `list` contenant les coefficients du polynôme

Pour construire le polynôme $4x^2 - 1$ on procède ainsi

```
>>> p1 = Polynome(2, [-1, 0, 4])
>>> p1.deg
2
>>> p1.coeff
[-1, 0, 4]
```

Ecrire le constructeur de la classe `Polynome`

2. Ecrire la méthode `somme(other)` qui renvoie un nouvel objet de type `Polynome`, somme de `self` et de `other`.

Par exemple la somme de $4x^2 - 1$ et de $3x^3 - 2x^2 + 2x + 1$ donne $3x^3 + 2x^2 + 2x$

```
>>> p1 = Polynome(2, [-1, 0, 4])
>>> p2 = Polynome(3, [1, 2, -2, 3])
>>> p3 = p1.somme(p2)
>>> p3.coeff
[0, 2, 2, 3]
>>> print(p3)
3x^3+2x^2+2x
```

3. Etant donné un polynôme $P(x) = \sum_{k=0}^{k=n} a_k x_k$, la **dérivée** de ce polynôme est définie par :

$$P'(x) = \sum_{k=1}^{k=n} k a_k x^{k-1}$$

Ecrire la méthode `derive` qui renvoie un nouvel objet de type `Polynome` la dérivée de `self`

```
>>> p1 = Polynome(2, [-1, 2, 4])
>>> p2 = p1.derive()
>>> print(p2)
8x+2
```

4. (Bonus) Evaluer un polynôme c'est donner une valeur à la variable x . Par exemple si le polynôme est

$3x^5 - 2x^4 + 5x^3 + 4x^2 - 2x + 1$ et si on remplace x par 1 on obtient après calcul 9.

Ecrire une méthode `eval(val)` qui renvoie un entier obtenu par calcul en remplaçant `x` par `val` dans `self`

5. (Bonus) Dans le but de diminuer le nombre de multiplications pour évaluer un polynôme on va mettre au point une autre méthode appelée **méthode de Horner**

Voici l'idée :

$$3x^5 - 2x^4 + 5x^3 + 4x^2 - 2x + 1 = (((((3x - 2)x + 5)x + 4)x - 2)x + 1$$

- (a) Ecrire une méthode `horner_1(val)` qui procède de manière **itérative**
- (b) Ecrire une méthode `horner_2(val)` qui procède de manière **réursive**

Ex 4

On demande de mettre au point la fonction `supprimer_2(liste:Cellule_2,adr:Cellule_2)` pour une liste doublement chaînée, où chaque cellule a un pointeur sur la cellule suivante et la cellule précédente.

la première cellule a son attribut `pred` qui vaut `None`, la dernière a son attribut `succ` qui vaut `None`.

`None` est représenté par une diagonale dans le schéma ci-dessous.

```
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:
    pass
```

Réfléchir à partir du schéma suivant représentant une liste contenant dans l'ordre les valeurs 1, 2 et 3 et où on veut supprimer la valeur 2 et on nous donne `adr` l'adresse mémoire de la cellule contenant 2.

Il faut donc raccorder la cellule contenant 1 et la cellule contenant 3 (voir les flèches en noir et en gras).

Contrairement à ce qui a été fait en classe ici on n'a pas besoin de rechercher le prédécesseur de `adr` avec une boucle `while`, il suffit de faire `adr.pred`.

Par contre avant d'écrire `adr.pred.succ` il faut s'assurer que `adr.pred` n'est pas `None`

