NSI Terminale S1 : Structures données linéaires et dictionnaires

Jusqu'à présent en première année, nous avons vu les données primitives comme par exemple les entiers, les flottants, les booléens (données disponibles dans les langages courants). Nous allons voir dans cette séquence, les données abstraites, c'est-à-dire des données non disponibles dans les langages de programmation courants. Nous allons étudier 4 types de structures de données abstraites :

- Les structures linéaires : les listes, les piles et les files.
- Les structures par accès par clé : les dictionnaires (déjà vu en première en en S3)
- Les structures hiérarchiques : les arbres
- Les structures relationnelles : les graphes.

Toutes ces structures possèdent un ensemble de routines (procédure ou fonctions) pour ajouter, effacer ou accéder aux données appelé interface.

1. Les listes :

a) Définition :

Une liste est une structure de données permettant de regrouper des données et dont l'accès est séquentiel. Elle correspond à une suite finie d'éléments repérés par leur rang(index).
ATTENTION : le type liste de Python ne correspond pas à la structure de données liste, le type « list » représente un tableau dynamique.

b) Les opérations :

Exemples :

- CREER_LISTE_VIDE(n) qui retourne un objet de type Liste de n éléments.
- INSERER(L,E,i) qui insère l'élément E à l'indice i dans la liste L si et seulement si 1≤i≤LONGUEUR(L).
- SUPPRIMER(L,i) qui supprime l'élément à l'indice i de la liste si et seulement si 1≤i≤LONGUEUR(L)+1.
- RECHERCHER(L,E) qui retourne un entier qui correspond à l'indice de l'élément trouvé , il renvoie -1 si non trouvé.
- INDEXER(L,i) qui retourne l'élément à l'indice i si et seulement si 1≤i≤LONGUEUR(L).
- MODIFIER(L,E,i) qui remplace l'élément à l'indice i par l'élément E si et seulement si 1≤i≤LONGUEUR(L).
- LONGUEUR(L) qui retourne un entier qui représente le nombre d'éléments dans le tableau.

c) Représentation :

On représente en général les listes par un tableau de taille fixe dont chaque élément est identifié par son indice.
On stockera à l'indice 0, le nombre d'éléments dans le tableau. Lorsqu'on créé la liste, si L[0]==0 alors la liste est vide.
Si L[0]==n+1 alors la liste est pleine

Exemple :

Liste de taille maximale 6 comportant 4 éléments : (3,7,1,8)

4	3	7	1	8		
0	1	2	3	4	5	6

Pour remplir ce tableau, la technique la plus utilisée sera :
INSERER(L,3,1)
INSERER(L,7,2)
INSERER(L,1,3)
INSERER(L,8,4)

I) D'après le pseudo-code ci-dessous, coder la fonction INSERER(L,E,i) :

```
Si (L[0]==len(L)) OU (i-1>L[0])
alors
        Afficher « La liste est pleine ou l'index i n'est pas correct »
        Retourner FAUX
Sinon
        Pour k allant de L[0]+1 à i+1 par pas de -1 :
            L[k]=L[k-1]
        L[i]=E
        L[0]=L[0]+1
        Retourner VRAI
```

Tester votre code en ajoutant après votre fonction :

```
def CREER_LISTE_VIDE(n):
    L=[None]*(n+1) #Création de liste vide
    L[0]=0 #On initialise la case 0 à 0 éléments
    return L

L=CREER_LISTE_VIDE(6)#On créé une liste vide de 6 éléments
print(L)
INSERER(L,3,1)
INSERER(L,7,2)
INSERER(L,1,3)
INSERER(L,8,4)
print(L)
```

Sauvegarder sous Exo1.py et capturer le résultat.

II) D'après le pseudo-code ci-dessous, coder la fonction SUPPRIMER(L,i) au bon endroit :

```
Si (L[0] !=0) ET (i<=L[0])
alors
        Pour k allant de i à L[0]-1 par pas de 1 :
            L[k]=L[k+1]
        Mettre à None l'élément que l'on supprime (celui à l'indice L[0])
        Décrémenter de 1 le nombre d'éléments dans la liste (L[0]=L[0]-1)
        Retourner VRAI
Sinon
        Afficher « La liste est vide ou l'index i n'est pas correct »
        Retourner FAUX
```

Tester votre code en ajoutant après le INSERER(L,8,4) : SUPPRIMER(L,2)
Sauvegarder sous Exo2.py et capturer le résultat.

III) Proposer un script pour la fonction RECHERCHER(L,E) , rajouter à la fin du programme :

```
print(RECHERCHER(L,1))
print(RECHERCHER (L,7))
```

Sauver sous Exo3.py et tester on attend sur la console :

```
[0, None, None, None, None, None, None]
[3, 3, 1, 8, 8, None, None]
2
-1
```

IV) Proposer un script pour la fonction INDEXER(L,i), rajouter à la fin du programme :

```
print(INDEXER(L,1))
print(INDEXER (L,4))
```

Sauver sous Exo4.py et tester on attend sur la console :

```
-1
3
Il n'y a pas d'élément à cet indice
```

V) Proposer un script pour la fonction MODIFIER (L,E,i), rajouter à la fin du programme :

```
MODIFIER(L,4,3)
MODIFIER(L,10,6)
print(L)
```

Sauver sous Exo5.py et tester on attend sur la console :

```
Il n'y a pas d'élément à cet indice à modifier
[3, 3, 1, 4, 8, None, None]
```

VI) Proposer un script pour la fonction LONGUEUR(L), rajouter à la fin du programme :

```
print(LONGUEUR(L))
```

Sauver sous Exo6.py et tester on attend sur la console :

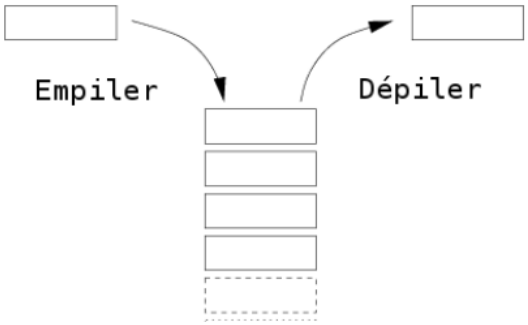
```
3
```

2. Les piles:

a) Définition :

Une pile est une structure de données qui donne accès en priorité aux dernières données ajoutées. On ne peut accéder qu'a l'objet situé au sommet de la pile. On décrit souvent se comportement « Dernier entré, premier sorti » en anglais LIFO « Last In, First Out »
Exemple de la vie courante : le rangement d'assiettes.

b) Les opérations :



Exemples :

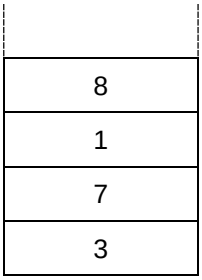
- CREER_PILE_VIDE(n) qui retourne une pile vide de n éléments.
- EMPILER(P,E) qui insère l'élément E au sommet de la pile si et seulement si P n'est pas pleine
- DEPILER(P) retourne la pile en supprimant l'élément au sommet de la pile si et seulement si P n'est pas vide.

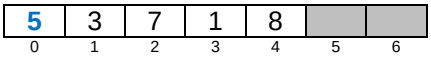
c) Représentation :

On représente en général une pile par un tableau de taille fixe dont chaque élément est identifié par son indice.
On stockera à l'indice 0, l'indice de la prochaine case vide (il correspondra à l'indice du prochain élément à insérer dans la pile)
Si P[0]==1 alors la pile est vide. A chaque fois qu'on insère un élément on augmente P[0].
SI P[0]==n+1 alors la pile est pleine.

Exemple :

Pile de taille maximale 6 comportant 4 éléments : (3,7,1,8)





Pour remplir cette pile, la technique utilisée sera :
EMPILER(Pile,3)
EMPILER(Pile,7)
EMPILER(Pile,1)
EMPILER(Pile,8)

VII) D'après le pseudo-code ci-dessous, coder la fonction EMPILER(P,E) :

```
Si (P[0]==len(P))
alors
        Afficher « La pile est pleine »
        Retourner FAUX
Sinon
        P[P[0]]=E
        P[0]= P[0]+1
        Retourner VRAI
```

A la suite de cette fonction, ajouter les lignes de code suivantes :

```
def CREER_PILE_VIDE(n):
    P=[None]*(n+1) #Création de liste vide
    P[0]=1 #On initialise la case 0 à l'indice du premier sommet 1
    return P

Pile=CREER_PILE_VIDE(6)#On créé une liste vide de 6 éléments
print(Pile)
EMPILER(Pile,3)
EMPILER(Pile,7)
EMPILER(Pile,1)
EMPILER(Pile,8)
print(Pile)
```

Sauvegarder sous Exo7.py et capturer le résultat.

VIII) D'après le pseudo-code ci-dessous, coder la fonction DEPILER(P) :

```
Si (P[0] !=1)
alors
        P[0]=P[0]-1
        Retourner P[P[0]]
Sinon
        Afficher « La pile est vide ! »
```

Tester votre code en ajoutant après la ligne EMPILER(Pile,8) : DEPILER(Pile)
Sauvegarder sous Exo8.py et capturer le résultat. Est-ce normal ?

3. Les files:

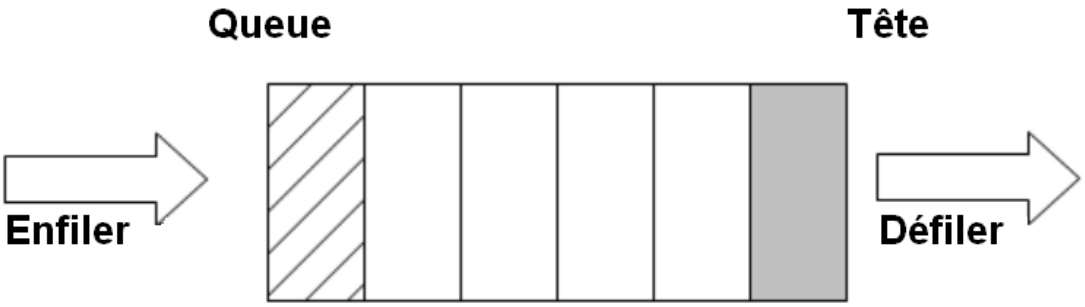
a) Définition :

Une file est une structure de données qui donne accès en priorité premier entré. On décrit souvent se comportement « Premier entré, premier sorti » en anglais FIFO « First In, First Out »
Exemple de la vie courante : Une file d'attente.

b) Les opérations :

Exemples :

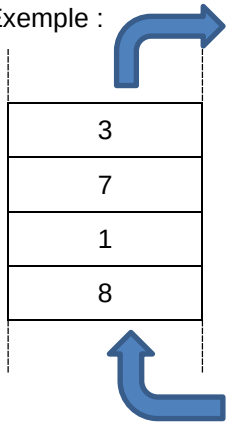
- CREER_FILE_VIDE(n) qui retourne une file vide de n éléments.
- ENFILER(F,E) qui insère l'élément E à la suite de la file si et seulement si F n'est pas pleine
- DEFILER(F) retourne la file en supprimant l'élément en tête de la file si et seulement si F n'est pas vide.



c) Représentation :

On représente en général une file par un tableau de taille fixe (n+2) dont chaque élément est identifié par son indice.
On stockera à l'indice 0, l'indice de la tête de la file
On stockera à l'indice 1, l'indice de la queue de la file
On stockera à l'indice 2, le nombre d'éléments de la file
On stockera dans les indices 3 à n+2 les éléments de la file ou vide
Si F[2]==0 alors la file est vide. A chaque fois qu'on enfile un élément on augmente F[2].
Si F[2]==n alors la file est pleine. A chaque fois qu'on défile un élément on diminue F[2].
Dès que les indices de tête ou de queue dépassent la longueur du tableau (n+3), ils repartent au début du tableau (On appelle cela un tableau avec une gestion circulaire)

Exemple :



File de taille maximale 6 comportant 4 éléments : (3,7,1,8)

Tête	Queue	Taille						
3	7	4	3	7	1	8		
0	1	2	3	4	5	6	7	8

Pour remplir cette file, la technique utilisée sera :
ENFILER(File,3)
ENFILER(File,7)
ENFILER(File,1)
ENFILER(File,8)

IX) D'après le pseudo-code ci-dessous, coder la fonction ENFILER(F,E) :

```
Si (F[2]==len(F)-3)
alors
        Afficher « La file est pleine »
        Retourner FAUX
Sinon
        F[F[1]]=E
        Si F[1]==len(F)-1 : # si on est sur le bout de la file on revient au début
                F[1]=3
        Sinon : #On ajoute 1 à la queue
                F[1]=F[1]+1
        F[2]=F[2]+1 #On ajoute 1 à la taille
        Retourner VRAI
```

A la suite de cette fonction, ajouter les lignes de code suivantes :

```
def CREER_FILE_VIDE(n):
   F=[None]*(n+3) #Création de file vide
   F[0]=3 #On initialise la case 0 à l'indice du premier élément soit 3
   F[1]=3 #On initialise la case 1 à l'indice du dernier élément ici 3 puisqu'il n'y en pas, elle est vide
   F[2]=0 #On initialise la case 2 à 0 qui correspond à la taille (0 élément).
   return F

File=CREER_FILE_VIDE(6)#On créé une File vide de 6 éléments
print(File)
ENFILER(File,3)
ENFILER(File,7)
ENFILER(File,1)
ENFILER(File,8)
print(File)
```

Sauvegarder sous Exo9.py et capturer le résultat.

X) D'après le pseudo-code ci-dessous, coder la fonction DEFILER(F) :

```
Si (F[2]==0) #si la taille =0
alors
        Afficher « La file est vide ! »
Sinon
        element=F[F[0]] #on mémorise l'élément à supprimer
        Si (F[0]==len(F)-1) : #On est à la fin de la file
                F[0]=3 #On revient au début de la file
        Sinon :
                F[0]=F[0]+1 #on décale la tête de 1
        F[2]=F[2]-1 #on retranche 1 à la taille de la file
        Retourner element
```

Tester votre code en ajoutant après la ligne ENFILER(File,8) : DEFILER(File)
Sauvegarder sous Exo10.py et capturer le résultat. Est-ce normal ?

4. Les dictionnaires : vus en première

a) Définition :

Un dictionnaire stocke des données sous la forme **clé ⇒ valeur.** Une clé est unique et n'est pas nécessairement un entier.
Exemple : les contacts d'un répertoire téléphonique chaque Nom est associé à un Numéro de téléphone.

b) Les opérations :

Exemple :

- Création d'un dictionnaire :
  francais_anglais = {"chien":"dog","chat":"cat","oiseau":"bird" }
- Ajout d'un élément :
  francais_anglais["souris"]="mouse"
- Suppression d'un élément :
  On utilise la méthode pop(clé)
  francais_anglais.pop("souris")

5. Exercices :

XI) Pour chacun des cas suivants, donner la structure de données à choisir :
a) Stockage de l'historique de pages web. (possibilité de revenir sur les pages précédentes à l'aide de la flèche ←).
b) Envoyer des pdf à imprimer sur le photocopieur du lycée.
c) Représenter un répertoire téléphonique.
d) Dans un logiciel, stocker les actions en vue d'utiliser l'action undo (annuler la dernière action faite).

XII) Décrypter le pseudo code suivant :

```
L1=CREER_LISTE_VIDE(5)
L2=CREER_LISTE_VIDE(5)
INSERER(L1,0,1)
INSERER(L1,1,2)
INSERER(L1,2,3)
INSERER(L1,3,4)
INSERER(L1,4,5)
INSERER(L2,INDEXER(L1,1),1)
INSERER(L2,INDEXER(L1,2),1)
INSERER(L2,INDEXER(L1,3),1)
INSERER(L2,INDEXER(L1,4),1)
INSERER(L2,INDEXER(L1,5),1)
```

Donner le contenu de L1 et L2. Que fait ce programme ? Coder, sauvegarder sous Exo12.py et tester afin de vérifier votre réponse.

XIII) Décrypter le pseudo code suivant :

```
Pile=CREER_PILE_VIDE(5)
EMPILER(Pile,'K')
EMPILER(Pile,'A')
EMPILER(Pile,'Y')
EMPILER(Pile,'A')
EMPILER(Pile,'K')
DEPILER(Pile)
DEPILER(Pile)
DEPILER(Pile)
DEPILER(Pile)
EMPILER(Pile,'O')
EMPILER(Pile,'*')
EMPILER(Pile,'*')
EMPILER(Pile,'*')
EMPILER(Pile,'*')
```

Donner le contenu de la pile. Que fait ce programme ? Coder, sauvegarder sous Exo13.py et tester afin de vérifier votre réponse.

XIV) Décrypter le pseudo code suivant :

```
File=CREER_FILE_VIDE(5)
ENFILER(File,'K')
ENFILER(File,'A')
ENFILER(File,'Y')
```

```
ENFILER(File,'A')
ENFILER(File,'K')
DEFILER(File)
DEFILER(File)
DEFILER(File)
DEFILER(File)
ENFILER(File,'*')
ENFILER(File,'*')
ENFILER(File,'*')
ENFILER(File,'O')
ENFILER(File,'*')
```

Donner le contenu de la pile. Que fait ce programme ? Coder, sauvegarder sous Exo14.py et tester afin de vérifier votre réponse.

Jusqu'à présent, nous avons créé des fonctions pour les piles et les files à l'aide d'instructions de base de python. Il existe, des méthodes à appliquer sur les tableaux append(), pop()… https://docs.python.org/fr/3/tutorial/datastructures.html

XV) A l'aide de celles-ci, créer un programme contenant les fonctions :

- CREER_PILE()
- EMPILER(p,e)
- DEPILER(p)

Tester votre code en ajoutant les lignes suivantes :

```
pile=CREER_PILE()
EMPILER(pile,'B')
EMPILER(pile,'O')
EMPILER(pile,'N')
EMPILER(pile,'J')
EMPILER(pile,'O')
EMPILER(pile,'U')
EMPILER(pile,'R')
EMPILER(pile,'!')
print(DEPILER(pile))
print(pile)
```

Sauvegarder sous Exo15.py

XVI) A l'aide des mêmes méthodes, créer un programme contenant les fonctions :

- CREER_FILE()
- ENFILER(f,e)
- DEFILER(f)

Tester votre code en ajoutant les lignes suivantes :

```
file=CREER_FILE()
ENFILER(file,'B')
ENFILER(file,'O')
ENFILER(file,'N')
ENFILER(file,'J')
ENFILER(file,'O')
ENFILER(file,'U')
ENFILER(file,'R')
ENFILER(file,'!')
print(DEFILER(file))
print(file)
```

Sauvegarder sous Exo16 .py

XVII) Le but de cet exercice est d'écrire une fonction qui contrôle si une expression mathématique, donnée sous forme d'une chaîne de caractères, est bien parenthésée, c'est-à-dire s'il y a autant de parenthèses ouvrantes que de fermantes.
Vous ne pouvez pas utiliser des variables qui "comptent" les parenthèses ouvrantes ou fermantes.

Exemples :

- (..(..)..) est bien parenthésée.
- (...(..(..)...) mal parenthésée .

Pour répondre à cet exercice, on va :
Créer une pile.
Créer une fonction def verification(expression) qui :
- Parcourir l'expression de gauche à droite.
- Si on a une parenthèse ouvrante "(" on l'empile.
- Si on a une parenthèse fermante ")" et que la pile n'est pas vide on dépile (sinon on retourne faux).
- À la fin la pile doit être vide et donc la fonction retourne vrai.

Sauver votre programme sous Exo17.py puis tester avec les expressions suivantes qui seront demandées à l'utilisateur à l'aide d'une commande input:
- "13*2+(5*6)+10/(5+1) "
- " (3*(2+1)/(15/6)+6 "
- " "

La Notation Polonaise Inverse (NPI), ou notation *post-fixée*, est une marnière d'écrire les expressions mathématiques en se passant des parenthèses. Elle a été introduite par le mathématicien polonais Jan Lucasievicz dans les années 1920.
Le principe de cette méthode est de placer chaque opérateur juste après ses deux opérandes. L'expression **2+3** devient en NPI **2 3 +.**
Autres exemples :

- 2+6−1 s'écrit 2 6 + 1 -
- 5∗3+4 s'écrit 5 3 * 4 +
- ((1+2)∗4)+3 s'écrit 1 2 + 4 * 3 +

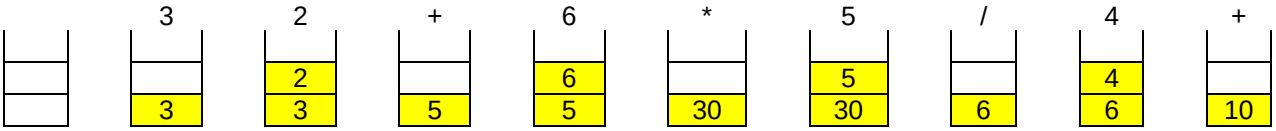
Évaluer une expression post-fixée est facile, il suffit de lire l'expression de gauche à droite et d'appliquer chaque opérateur aux deux opérandes qui le précèdent. Si l'opérateur n'est pas le dernier symbole on replace le résultat intermédiaire dans l'expression et on recommence avec l'opérateur suivant.

Le but de l'exercice est de réaliser en Python une calculatrice simple, capable d'évaluer une formule en NPI et de retourner le résultat arithmétique. La réalisation d'une telle calculatrice se fera à l'aide d'une pile.

Algorithme :

- Lire un par un les caractères de l'expression.
- Si le caractère lu est un opérande alors on l'empile.
- Si le caractère lu est un opérateur, alors on dépile les deux éléments se trouvant en haut de la pile, on calcule le résultat en appliquant l'opérateur sur les deux opérandes dépilés et on empile le résultat.
- Une fois tous les caractères lus, la pile ne contient qu'un seul élément qui correspond au résultat final.

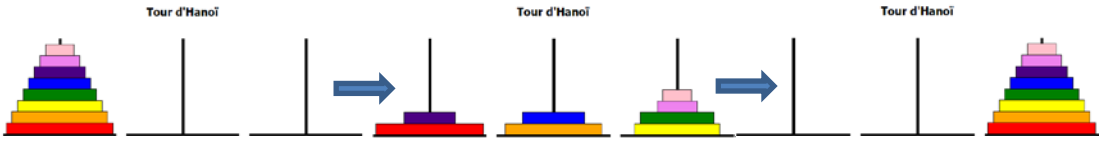
Exemple :
((3+2)∗6)/5+4, ou 3 2 + 6 * 5 / 4 + en NPI.



XVIII) Le but de cet exercice est d'écrire un programme qui donne le résultat du calcul lorsqu'on donne celui sous forme de notation polonaise inversée. Pour cela, vous allez créer une fonction qui va permettre de faire le calcul de chaque opération : def calcul(operation,x,y) qui retournera le calcul de x operation y (où opération est +,*,- ou /)
Ensuite créer une fonction def evaluation(expresion) qui :
- Créer une pile vide
- Pour chaque caractère, on teste s'il est différent de + et * et / et – alors on empile celui-ci dans la pile SINON on convertit les caractères x et y de la pile en float après les avoir dépilés puis on calcule le résultat de l'opération à l'aide de la fonction calcul et on empile celui-ci après l'avoir converti en chaine.
- A la fin on renvoie le résultat.

La chaine à calculer sera demandé à l'utilisateur à l'aide d'une fonction input et on affichera le résultat du calcul.
Sauvegarder sous Exo18.py et tester avec l'expression : 32+6*5/4+

Les tours de Hanoï :



On dispose de 3 piles p1,p2 et p3. Dans la pile p1 sont empilés dans l'ordre croissant les nombres n,n-1,n-2,….1. Les piles p2 et p3 sont vides.
L'objectif est d'obtenir une pile p3 identique à la pile p1 initiale en un minimum de coups, les piles p1 et p2 sont vides.
Un coup est le déplacement d'un nombre, sommet d'une pile, sur une autre pile. Pour ce faire, nous allons utiliser les fonctions EMPILER et DEPILER.
On empile un nombre sur une pile uniquement si la pile est vide ou si le nombre est plus petit que le sommet de la pile.
Pour résoudre se problème, on utilise la récursivité :

- Déplacer de la tour p1 vers la tour p2 les n-1 premiers disques (dep=p1, arr=p2, inter=p3)
- Déplacer le plus grand disque de tour p1 vers la tour p3
- Déplacer de la tour p2 vers p3 les n-1 premiers disques. (dep=p2, arr=p3, inter=p1)

XIX) Le but de cet exercice est d'écrire un programme qui affiche chaque coup et qui les compte.
Nous allons créer la fonction def Hanoi(n,depart,arrivee,intermediaire) que déplacera les disque le la tour de départ (p1) vers l'arrivée(p3) en passant par l'intermédiaire(p2). On incrémentera un compteur (variable globale) qui sera initialiser à 0 en début de programme.
Pour empiler p1 on utilisera une boucle for allant de n à 1 avec un incrément de -1.
On affichera les 3 piles avant et après pour vérifier le bon fonctionnement ainsi que la valeur du compteur qui représente le nombre de coups.
Sauvegarder sous Exo19.py et tester avec :

- n=3 et comparer la valeur du compteur avec 2^3-1,
- n=8 et comparer la valeur du compteur avec 2^8-1, puis
- n=10 et comparer la valeur du compteur avec 2^{10}-1.

D'après vous combien de coups faut-il pour déplacer 32 disques ?
Sachant que l'on met au moins 1s à déplacer un disque, pensez-vous pouvoir résoudre le jeu avec 32 disques sans ordinateur(justifiez) ?