La récursivité

Langages et programmation

Contenu : Récursivité.
Capacités attendues : Écrire un programme récursif.
Analyser le fonctionnement d'un programme récursif.

I. Notion de récursivité - 1

Écrire une procédure récursive ***nbre_positif()*** qui demande un nombre positif à l'utilisateur et qui affichera ce nombre que s'il est positif. Sinon, demander à l'utilisateur de recommencer.

II. Notion de récursivité - 2

Vous avez déjà beaucoup travaillé avec la fonction ***puissance*** qui renvoie x^n.

```
def puissance (x : float, n : int) -> float:
    if n==0 :
        return 1
    else :
        return x*puissance_rec(x, n-1)
```

Le but est de créer une fonction récursive ***somme_puissance_rec*** qui permet d'obtenir la somme $1 + x^1 + x^2 + ... + x^{n-1} + x^n$, sachant que x, un réel, et n , un entier naturel, sont deux arguments de cette fonction.
Créer une telle fonction ***somme_puissance_rec*** qui est récursive et qui peut aussi faire appel à la fonction ***puissance_rec***.

III. Notion de récursivité - 3

Écrire une procédure récursive qui affiche un triangle rempli d'étoiles (*) sur un nombre de lignes donné passé en paramètre, exemple avec 6 lignes:

```
******
*****
****
***
**
*
```

On demandera à l'utilisateur combien de lignes il souhaite.

IV. Notion de récursivité - 4

Une espèce de papillon est désormais considérée comme menacée en France. L'espace protégé où il vit, cherche à y maintenir, voire à y développer la population grâce à un dispositif mis en place à partir de 2020.
En 2020, la population de fadets des tourbières est estimée à 150 individus dans l'espace protégé. On note ***nb_fadet(n)*** la population de papillon de cette espèce vivant dans l'espace protégé n années après 2020. Les écologues responsables du dispositif espèrent que chaque année 80% de la population précédente reste et que 50 individus s'installent dans l'espace protégé.
On peut donc écrire cette relation ainsi : **nb_fadet(n+1) = 0,8 × nb_fadet(n) + 50**

1. En admettant cette relation, proposer une fonction ***recursive nb_fadet*** qui prend comme argument l'entier naturel n et renvoie la population ***nb_fadet(n)*** estimée pour l'année **2020 + n**.
2. Au vu du modèle, peut-on espérer une sauvegarde de l'espèce dans la zone protégée ?

V. Notion de récursivité - 5

Écrire une fonction récursive ***inverse()*** qui prend en paramètre une chaine de caractères **chaine** et renvoie la chaine obtenue en inversant l'ordre des caractères. Par exemple, ***inverse("azerty")*** a pour valeur la chaine "*ytreza*". On pourra utiliser le fait que *chaine[1:]* correspond à la chaîne de caractères **chaine** privée du premier caractère, chaîne éventuellement vide.

VI. Pile d'exécution

Le mathématicien Lothar Collatz a inventé il y a près d'un siècle la suite mathématique suivante : Prendre un nombre entier au hasard. S'il est pair le diviser par 2, sinon, le multiplier par 3 et ajoutez 1. Répéter ensuite l'opération.
Cette suite peut être écrite mathématiquement par la **relation de récurrence** suivante :un+1={un2,si n est pair.3×un+1,si n est impair.un+1={un2,si n est pair.3×un+1,si n est impair.
Collatz puis de nombreux mathématiciens ont cherché à prouver que quel que soit le nombre de départ choisi, on arrivera à la valeur 1 donc à la répétition infinie 4, 2, 1. Ce résultat non encore démontré est appelée conjecture de Syracuse

Cette conjecture mobilisa tant les mathématiciens durant les années 1960, en pleine guerre froide, qu'une plaisanterie courut selon laquelle ce problème faisait partie d'un complot soviétique visant à ralentir la recherche américaine.

1. Proposez une fonction récursive syracuse qui prend en paramètre un entier naturel n et qui renvoie la liste de toutes les valeurs prises par le suite jusqu'à atteindre la valeur 1.
   - Tester votre fonction Syracuse avec n=15 entre autres et vous devez obtenir l'affichage suivant :

```
>>>syracuse(15)
[15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

   - Vous pourrez utilisez la méthode ***extend*** : ***liste1.extend(liste2)*** permet de rajouter en fin de *liste1* tous les éléments de la liste *liste2*.
2. On appelle **temps de vol** le plus petit indice p tel que up=1.
   Proposer une fonction ***temps_vol*** qui prend en paramètre un entier n et qui renvoie le premier indice où la suite de Syracuse prend la valeur 1 si on part de la valeur n.
   Tester votre fonction ***temps_vol*** avec différentes valeurs de n puis avec le nombre 12235060455 ; que se passe-t-il dans ce dernier cas ?

VII. Notion de récursivité - 6

Pour convertir un nombre entier positif n de la base décimale à la base binaire, il suffit d'effectuer des divisions successives du nombre n par 2. La liste des restes des divisions constitue la représentation binaire.
Écrire une fonction récursive ***dec_vers_bin*** renvoyant la liste donnant une représentation binaire d'un nombre n saisi comme argument.

VIII. Notion de récursivité - 7

Rappel !
La factorielle d'un entier naturel n non nul, notée n!, est le produit de tous les nombres entiers compris entre 1 et n, c'est-à-dire :
n! = 1 × 2 × ... × (n−1) × n
Exemples : 4! = 1 × 2 × 3 × 4 = 24, 6! = 1 × 2 × 3 × 4 × 5 × 6 = 720 et 1! = 1.

Il existe une relation "simple" entre n! et (n−1)!
En effet : n! = 1 × 2 × ... × (n−1) × n = 1 × 2 × ... × (n−1) (n−1)! × n = (n−1)! × n
En sciences, la fonction exponentielle est très importante. On admet que la valeur de cette fonction en 1 est notée e^1 et que sa valeur peut être vue comme égale à la somme infinie : $1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!}$ où n va tendre vers +∞.

1. Écrire une fonction **récursive *inv_fact*** qui prend comme argument l'entier non nul n et qui renvoie $\frac{1}{n!}$:
2. Quelle est la condition d'arrêt de votre fonction récursive ?
3. Pourquoi peut-on être certain que la situation de terminaison sera atteinte après un nombre fini d'appels récursifs ?
4. Écrire une procédure qui affiche une valeur approchée de e^1 sachant que $e^1 \cong 1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{25!}$

IX. Calcul de chemin

On considère un tableau de nombres de *n* lignes et *p* colonnes.
Les lignes sont numérotées de 0 à *n* - 1 et les colonnes sont numérotées de 0 à *p* - 1.
La case en haut à gauche est repérée par (0*;* 0) et la case en bas à droite par (*n* - 1*; p* - 1).
On appelle *chemin* une succession de cases allant de la case (0*;* 0) à la case (*n* - 1*; p* - 1), en n'autorisant que des déplacements case par case : soit vers la droite, soit vers le bas.
On appelle *somme* d'un chemin la somme des entiers situés sur ce chemin.
Par exemple, pour le tableau T suivant :

4	**1**	**1**	3
2	0	**2**	1
3	1	**5**	**1**

- Un chemin est (0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3) (en gras sur le tableau)
- La somme du chemin précédent est 14
- (0, 0), (0, 2), (2, 2), (2, 3) n'est pas un chemin

L'objectif de cet exercice est de déterminer la somme maximale pour tous les chemins possibles allant de la case (0*;* 0) à la case (*n* - 1*; p* - 1).

1. On considère tous les chemins allant de la case (0, 0) à la case (2, 3) du tableau T donné en exemple.
   a. Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il ?

   b. La longueur d'un chemin est égal au nombre de cases de ce chemin. Justifier que tous les chemins allant de (0, 0) à (2, 3) ont une longueur égale à 6.

2. En listant tous les chemins possibles allant de (0, 0) à (2, 3) du tableau T, déterminer un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.

3. On veut créer le tableau T' où chaque élément T'[i][j] est la somme maximale pour tous les chemins possibles allant de (0, 0) à (I, j).

a. Compléter et recopier sur votre copie le tableau T' donné ci-dessous associé au tableau T =

4	1	1	3
2	0	2	1
3	1	5	1

T' =

4	5	6	?
6	?	8	10
9	10	?	16

b. Justifier que si j est différent de 0, alors : T' [0] [j] = T [0] [j] + T' [0] [j-1]

4. Justifier que si i et j sont différents de 0, alors : T' [i] [j] = T [i] [j] + max(T' [i-1] [j], T' [i] [j-1])

5. On veut créer la fonction récursive ***somme_max*** ayant pour paramètres un tableau T, un entier i et un entier j. Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case (0, 0) à la case (i, j).

a. Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la function somme_max ? Que renvoie-t-on dans ce cas ?

b. À l'aide de la question précédente, écrire en Python la fonction récursive ***somme_max*** dans pyzo ou un notebook..

c. Quel appel de fonction doit-on faire pour résoudre le problème initial ?

X. Tours de Hanoï

De nombreux jeu peuvent être résolu grâce à des algorithmes récursifs. Un des exemples courants utilisant la récursivité est le casse-tête des tours de Hanoï.

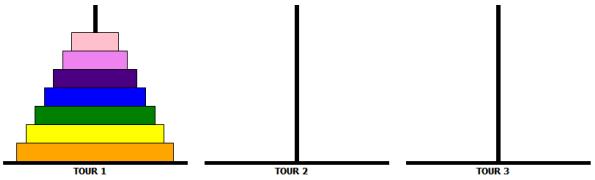
Ce casse-tête est composé de trois tours et une pile de disques rangés du plus grand au plus petit. Les disques sont initialement empilés à gauche. Il faut réussir à déplacer cette pile entièrement sur la tour de droite.

Pour cela, il faut respecter les règles suivantes :

- ne déplacer qu'un seul disque à la fois,
- un disque ne peut pas être posé sur un disque plus petit.

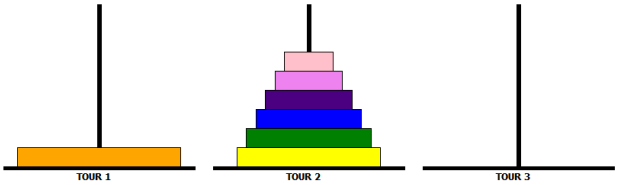
Afin de mieux comprendre le casse-tête, voici une animation venant de Wikipédia dans le cas où il y a 3 disques.

Notons **TOUR_1**, **TOUR_2** et **TOUR_3** les trois emplacements des tours, **TOUR_1** étant celle de gauche par exemple.

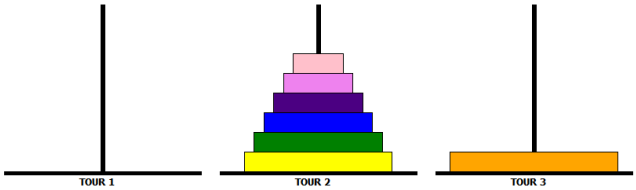


Pour déplacer une tour de n disques de **TOUR_1** vers **TOUR_3**, on effectue ces trois étapes :

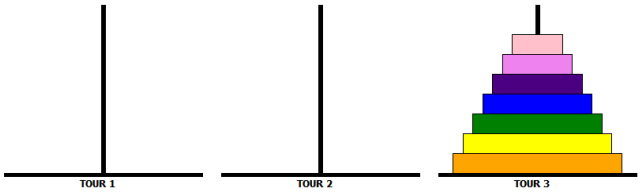
- déplacer la tour des n−1 premiers disques de **TOUR_1** vers **TOUR_2** ;



- déplacer le plus grand disque de **TOUR_1** vers **TOUR_3** ;



- déplacer la tour des n−1 premiers disques de **TOUR_2** vers **TOUR_3**.



Créer une fonction récursive ***hanoi(n,depart,inter,arrivee)*** où :

- n est le nombre de disques à déplacer,
- *depart* est la tour de départ ayant n disques,
- *inter* est la tour intermédiaire que l'on peut utiliser pour déplacer,
- *arrivee* est la tour ou doivent se trouver les n disques finalement.

Cette fonction renverra une succession d'affichage expliquant à chaque fois la position de la pièce à déplacer puis la position où elle doit être déplacée. La deuxième étape correspondra à un affichage du type : print ("Déplacer ",...,"sur",...). on peut remarquer que lorsque n=0, il n'y a rien à faire.

En exécutant hanoi(3,"TOUR_1","TOUR_2","TOUR_3"), vous devez obtenir la sortie suivante:

Déplacer TOUR_1 sur TOUR_3
Déplacer TOUR_1 sur TOUR_2
Déplacer TOUR_3 sur TOUR_2
Déplacer TOUR_1 sur TOUR_3
Déplacer TOUR_2 sur TOUR_1
Déplacer TOUR_2 sur TOUR_3
Déplacer TOUR_1 sur TOUR_3

Ce casse-tête a été inventé par le mathématicien français Édouard Lucas. Afin de mettre en valeur son jeu, il a aussi inventé une histoire fabuleuse. Il prétendit que dans le grand temple de Bénarès en Inde, centre du monde pour les hindouistes, ce dispositif du casse-tête était présent : Trois aiguilles de diamant y seraient plantées dans une dalle d'airain. Sur une de ces aiguilles, le dieu Brahma enfila au commencement des siècles, 64 disques d'or pur, le plus large reposant sur l'airain, et les autres, de plus en plus étroits, superposés jusqu'au sommet. Nuit et jour, des prêtres se succèdent pour transporter la tour de la première aiguille sur la troisième, en respectant les règles fixes vues pour le casse-tête qui auraient été imposées par Brahma. Quand tout sera fini ce sera la fin des mondes.

Ne bramez pas : "La fin du monde !" Rassurez-vous, à raison d'un déplacement par seconde, on peut montrer mathématiquement, qu'il faudrait au moins 584 milliards d'années pour pouvoir déplacer la tour de 64 disques.

L'algorithme récursif précédent est très court et assez aisé à comprendre pourquoi il fonctionne.

Il est possible d'écrire des algorithmes itératifs qui affichent aussi la succession des déplacements à effectuer. Cependant, leur écriture est plus compliquée, plus longue et il est plus difficile à comprendre pourquoi ils fonctionnent correctement.