TP5: récursivité, chaînes de caractères

Le document de référence doit être apporté en TP. Les exercices 1 à 3, incontournables, doivent être parfaitement traités en séance de TP. Les exercices suivants peuvent être traités en fin de séance ou à la maison, pour s'entraîner. Tous ces exercices sont susceptibles d'être posés en DS.

Créer un dossier "TP5" dans votre répertoire personnel. Pour l'exercice 1, dans Pyzo, enregistrer un fichier nommé "exo1.py" dans le dossier "TP5". Pour l'exercice 2, dans Pyzo, enregistrer un fichier nommé "exo2.py" dans le dossier "TP5"..... Pour chaque exercice, le fichier python sera régulièrement sauvegardé et éxécuté intégralement avec la commande "Run file as script" du menu Run (Ctrl+Shift+E). Les instructions d'affichage seront saisies sur le fichier python. Au fur et à mesure de l'avancée de l'exercice, les instructions devenant inutiles et gênantes pourront être désactivées en utilisant le caractère #.

Exemple.

Donnée : n (un entier naturel).
Calcul récursif de $n!$:
Premier cas : $n = 0$. Le calcul de $n!$ retourne 1 (car $0! = 1$)
Second cas : $n \geq 1$. Le calcul de $n!$ retourne le résultat du calcul de $(n-1)!$ multiplié par n (car $n! = (n-1)! \times n$).

Fonction récursive "facto" ayant pour argument n (entier naturel) et retournant $n!$:

```
def facto(n) :
    if n==0 :
        return(1)
    else :
        return(facto(n-1)*n)
```

Exercice 1 (exponentiations)**.**

Données *: x (un nombre) et n (un entier naturel).*
Calcul récursif de x^n ***naïf*** *:*
Premier cas : $n = 0$. Le calcul naïf de x^n retourne 1 (car $x^0 = 1$).
Second cas : $n = 1$. Le calcul naïf de x^n retourne x (car $x^1 = x$).
Troisième cas : $n \geq 2$. Le calcul naïf de x^n retourne le résultat du calcul naïf de x^{n-1} multiplié par x (car $x^n = x^{n-1} \times x$)

Question 1. *Ecrire une fonction récursive "exponaive" ayant pour arguments x (nombre) et n (entier naturel) et retournant x^n calculé naïvement. Faire afficher $exponaive(2, 10)$.*

Données *: x (un nombre) et n (un entier naturel).*
Calcul récursif de x^n ***rapide*** *:*
Premier cas : $n = 0$. Le calcul rapide de x^n retourne 1 (car $x^0 = 1$).
Second cas : $n = 1$. Le calcul rapide de x^n retourne x (car $x^1 = x$).
Toisième cas : $n \geq 2$. On pose $q = n//2$ et $y = x \times x$.
Premier sous-cas : n pair. Le calcul rapide de x^n retourne le résultat du calcul rapide de y^q

(car $x^n = x^{2q} = (x^2)^q = y^q$).
Second sous-cas : n impair. Le calcul rapide de x^n retourne le résultat du calcul rapide de y^p multiplié par x (car $x^n = x^{2q+1} = (x^2)^q \times x = y^q \times x$).

Question 2. *Ecrire une fonction récursive "exporapide" ayant pour arguments x (nombre) et n (entier naturel) et retournant x^n calculé par rapidemment. Faire afficher exporapide$(2, 10)$.*

Remarque.

Pour tout $n \in \mathbb{N}^*$, le calcul naïf de x^n nécessite $n-1$ multiplications. Pour tout $n \in \mathbb{N}$, notons $f(n)$ le nombre de multiplications nécessaires au calcul rapide de x^n. $f(1) = 0$ et, pour tout $q \in \mathbb{N}$, $f(2q) = 1 + f(q)$ (car pour calculer x^{2q}, il faut 1 multiplication pour calculer $y = x \times x$ puis $f(q)$ multiplications pour calculer y^q). Donc on voit par récurrence que pour tout $p \in \mathbb{N}$, $f(2^p) = p$ donc, en posant $n = 2^p$, $f(n) = log_2(n)$. On peut montrer que, pour tout entier n suffisamment grand, $f(n)$ est de l'ordre de grandeur de $log_2(n)$ donc $f(n)$ est beaucoup plus petit que $n-1$ donc le calcul rapide de x^n nécessite beaucoup moins de multiplications que le calcul naïf de x^n.

Exercice 2 (Recherche dichotomique)**.**
Dans cet exercice, le terme tableau désigne une liste.

Données :
t un tableau de nombres trié
 # Pour tout indices i et j, $i \leq j \Rightarrow t[i] \leq t[j]$ donc, par contraposée, $t[i] > t[j] \Rightarrow i > j$
$[[a, b]]$ un intervalle d'indices
 # $[[a, b]]$ est un intervalle d'entiers inclus dans l'ensemble des indices
x un nombre.

Problème :
Rechercher x parmi $t[a], ..., t[b]$ et retourner un indice $k \in [[a, b]]$ tel que $t[k] = x$ si il existe un tel indice, et retourner None sinon.

Algorithme récursif de recherche dichotomique :
Premier cas : $a \leq b$.
 # Idée de la dichotomie : découper $[[a, b]]$ en deux moitiés
Considèrons $m = (a + b)//2$
 # m indice au plus proche du milieu de a et b
Premier sous-cas : $x < t[m]$.
La recherche dichotomique de x parmi $t[a], ..., t[b]$ retourne :
le résultat de la recherche dichotomique de x parmi $t[a], ..., t[m-1]$.
 # car pour tout $k \in [[a, b]]$, $t[k] = x \Rightarrow k \in [[a, m-1]]$
 # car $t[k] = x \Rightarrow t[k] < t[m] \Rightarrow k < m$
Second sous-cas : $t[m] < x$.
La recherche dichotomique de x parmi $t[a], ..., t[b]$ retourne :
le résultat de la recherche dichotomique de x parmi $t[m+1], ..., t[b]$
 # car pour tout $k \in [[a, b]]$, $t[k] = x \Rightarrow k \in [[m+1, b]]$
 # car $t[k] = x \Rightarrow t[m] < t[k] \Rightarrow m < k$
Troisième sous-cas : $t[m] = x$.
La recherche dichotomique de x parmi $t[a], ..., t[b]$ retourne m.
Second cas : $b > a$.
La recherche dichotomique de x parmi $t[a], ..., t[b]$ retourne None
 # car $[[a, b]]$ est vide.

Question 1. *Ecrire une fonction récursive nommée "rechdicho" ayant pour arguments t (tableau de nombres trié), a et b (entiers tels que $[[a, b]]$ est un intervalle d'indices) et x (nombre) et retournant un indice $k \in [[a, b]]$ tel que $t[k] = x$ si il existe un tel indice, et retournant None sinon.*

Question 2. *Ecrire une fonction nommée "rechdichoprinc" ayant pour arguments t (tableau de nombres trié) et x (nombre) et retournant un indice k tel que $t[k] = x$ si il existe un tel indice, et retournant None sinon. Cette fonction devra appeler la fonction "rechdicho". Faire afficher rechdichoprinc$([0, 2, 6, 12, 15, 19, 25, 32, 40], 7)$ et rechdichoprinc$([0, 2, 6, 12, 15, 19, 25, 32, 40], 25)$.*

Question 3. *On note $f(n)$ le nombre d'opérations $//2$ effectuées pour une recherche dichotomique dans un tableau de longueur n, dans le pire des cas (ie au maximum). On voit que $f(1) = 1$ et pour tout $n \in \mathbb{N}$, $f(2n) = 1 + f(n)$ (car pour une recherche dichotomique dans un tableau de longueur $2n$, 1 opération $//2$ permet de se ramener dans le pire des cas à une recherche dichotomique dans un tableau de longueur n).*

1. *Déterminer $f(2)$, $f(2^2)$, $f(2^3)$, $f(2^4)$.*
2. *Soit $p \in \mathbb{N}$. Exprimer $f(2^p)$ en fonction de p, sans justification.*
3. *On considère $n = 2^p$. Exprimer $f(n)$ en fonction de n.*

Remarque.

Dans le pire des cas, le nombre d'opérations nécessaires à effectuer une recherche dichotomique dans un tableau de longueur n est de l'ordre de $log_2(n)$. On dit que la complexité dans le pire des cas est logarithmique en ordre de grandeur.

Remarque.

Pour n grand, $log_2(n)$ est beaucoup plus petit que n. Donc, pour un grand tableau de nombres trié, l'algorithme de recherche dichotomique est beaucoup plus efficace (i.e. beaucoup moins couteux en opérations) que l'algorithme de recherche séquentielle.

Question 4. *Ecrire une fonction itérative nommée "rechdichoite" ayant pour arguments t (tableau de nombres trié) et x (nombre) et retournant un indice k tel que $t[k] = x$ si il existe un tel indice, et retournant None sinon. Cette fonction utilisera des variables a et b et une boucle "Tant que $a \leq b$"' de sorte que la propriété I :"$[[a, b]]$ est un intervalle d'indices et pour tout indice k, $t[k] = x \Rightarrow k \in [[a, b]]$" est un invariant de boucle. Faire afficher rechdichoite$([0, 2, 6, 12, 15, 19, 25, 32, 40], 7)$ et rechdichoite$([0, 2, 6, 12, 15, 19, 25, 32, 40], 25)$.*

Question 5. *Donner un variant de la boucle "Tant que $a \leq b$", sans justifier. Que peut-on en déduire ?*

Exercice 3. *Recopier et faire éxécuter la suite d'instructions suivante et examiner les affichages afin de s'assurer d'avoir compris le fonctionnement.*

```
c = "Il est grand"
print(c)
n = len(c)
print(c)
print(c[0])
print(c[1])
print(c[2])
print(c[n - 1])
```

Exercice 4. *Ecrire une fonction "comptage" ayant pour arguments une chaîne de caractères c et un caractère d et retournant le nombre de caractères dans c qui sont égaux à d.*
Faire afficher comptage("Ceci est un essai.","e").

Soit c une chaine de caractère. On appelle sous-chaîne de c toute chaine formée de caractères $c[i], c[i + 1], ..., c[k]$ avec i et k des indices de c tels que $i \leq k$.

Soit i un indice de c. On appelle sous-chaîne de c débutant à l'indice i toute chaine de caractères $c[i], c[i + 1], ..., c[k]$ avec k un indice de c tels que $i \leq k$.

Exercice 5 (Recherche d'un facteur dans un texte)**.**

1. *Ecrire une fonction rechposmot(c, i, m) ayant pour arguments c (chaine de caractères), i (indice de c) et m (chaine de caractère telle que $i + p \leq n$ où p et n sont les longueurs de m et c) et retournant True ssi m est une sous-chaîne de c débutant à l'indice i, et False sinon. Faire afficher rechposmot("Il est grand",4,"est") et rechposmot("Il est grand",3,"est").*
2. *Ecrire une fonction rechmot(c, m) ayant arguments c et m (chaînes de caractères) et retournant True si m est une sous-chaîne de c, et False sinon. rechmot ne devra pas faire appel à rechposmot, mais au lieu de cela on copiera la majeure partie du corps de rechposmot dans le corps de rechmot. Faire afficher rechmot("Il est grand","est") et rechmot("Il est grand","test").*

Exercice 6 (Tours de Hanoï)**.** *On dispose d'assiettes de largeurs 2 à 2 distinctes. On appelle tour de Hanoï toute pile d'assiettes dont toute assiette, sauf la plus basse, est moins large que l'assiette sur laquelle elle est posée. Une tour de Hanoï est représentée par la liste dont les termes sont les largeurs des assiettes de la pile disposées par ordre décroissant de sorte que le premier terme de la liste est la largeur de l'assiette la plus basse et le dernier terme est la largeur de l'assiette la plus haute.*

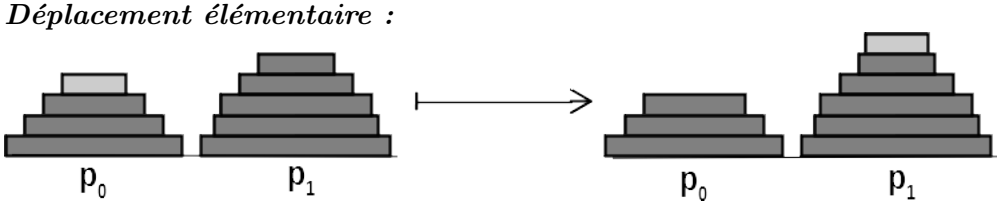
Question 1. *Affecter aux variables a et b les listes $[5, 3, 1]$ et $[9, 7, 4, 2]$ qui représentent des tours de Hanoï. Faire afficher a et b.*

Si l est une liste, $x = l.pop()$ permet de supprimer le dernier terme de la liste l tout en affectant à x la valeur de ce terme.

Question 2. *Recopier et faire éxécuter la suite d'instructions suivante et examiner les affichages afin de s'assurer d'avoir compris le fonctionnement.*

```
l = [12, 9, 6, 3, 2, 1, 0]
print(l)
x = l.pop()
print(l)
print(x)
```

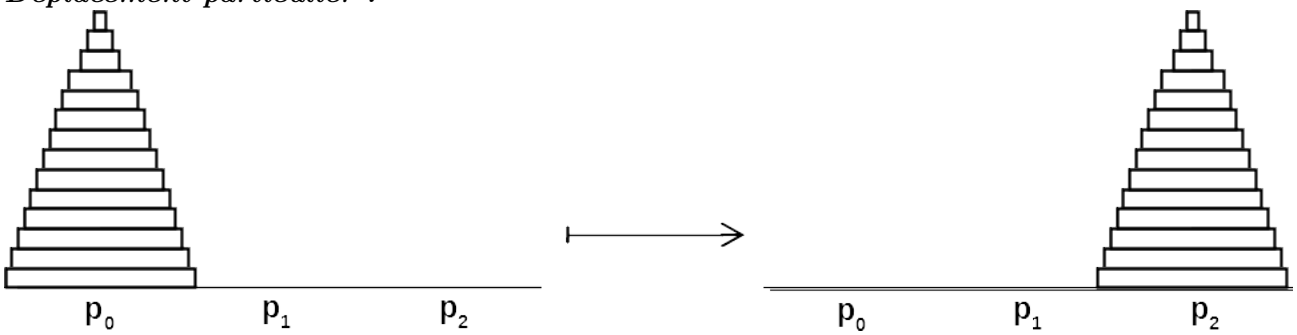
Déplacement élémentaire :



Question 3. *Ecrire une fonction "de" ayant prenant en arguments a, b (deux listes représentant des tours de Hanoï p_0 et p_1 telle que l'assiette la plus haute de p_0 est moins large que l'assiette la plus haute de p_1) et qui enlève l'assiette en haut de la pile p_0 puis la remet au dessus de la pile p_1 (effet de bord). Exécuter l'instruction de(a, b). Faire afficher a et b.*

On appelle déplacement de certaines d'assiettes sur une pile toute suite de déplacements élémentaires ne concernant que ces assiettes à l'issue de laquelle ces assiettes se retrouvent sur la pile.

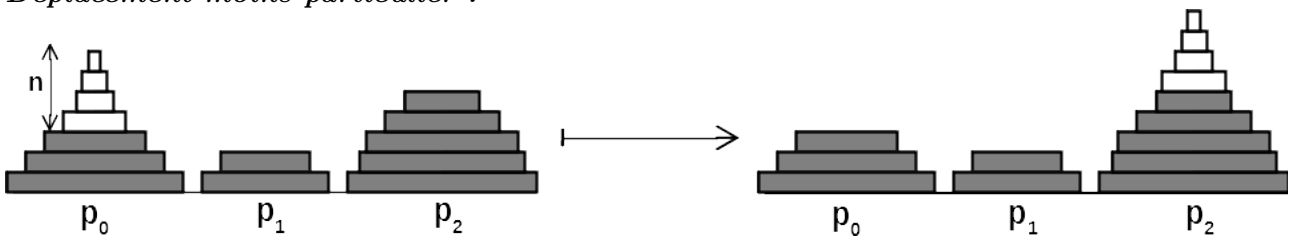
Déplacement particulier :



On souhaite définir une fonction nommée "dp" ayant pour argument a, b, c (des listes représentant des tours de Hanoï p_0,p_1,p_2 avec p_1 et p_2 vides) et déplacant toutes les assiettes de p_0 sur p_2 (effet de bord).

Pour atteindre cet objectif, on va considérer un déplacement moins particulier.

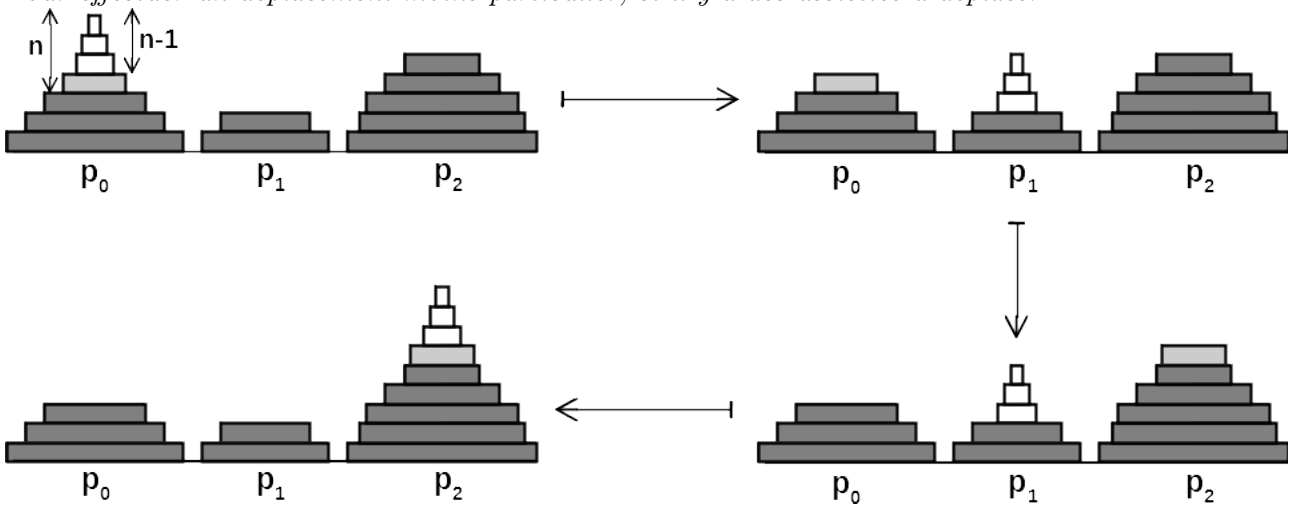
Déplacement moins particulier :



On souhaite définir une fonction nommée "dmp" ayant pour arguments a, b, c (des listes représentant des tours de Hanoï p_0,p_1,p_2) et n (entier naturel tel que p_0 possède au moins n assiettes et les n assiettes les plus hautes de p_0 sont moins larges que les assiettes de p_1 et p_2) et déplacant les n assiettes les plus hautes de p_0 sur p_2 (effet de bord)

Définition récursive du déplacement moins particulier :

Pour effectuer un déplacement moins particulier, si il y a des assiettes à déplacer :



Question 4. *Ecrire la fonction récursive "dmp". La fonction "dmp" devra faire appel à la fonction "de". Puis écrire la fonction "dp". Affecter auc variables k,l,m les listes $[3, 2, 1, 0]$, [], [] qui représentent des tours de Hanoï p_0,p_1,p_2 avec p_1 et p_2 vides. Affecter à la variable tours la valeur $[k, l, m]$. Faire afficher tours. Remarque importante : $[k, l, m]$ est la liste des références aux valeurs de k, l, m de sorte que si les listes k, l, m sont modifiées alors tours le sera également. Afin de visualiser, ajouter l'instruction print(tours) dans le corps de la fonction "dmp" juste après l'appel à la fonction "de". Faire éxécuter $dp(i, j, k)$.*

Déplacement général :

On souhaite écrire une fonction dg ayant pour arguments a,b,c (listes représentant des tours de Hanoï p_0,p_1,p_2) et déplacant toutes les assiettes sur la pile p_2 (effet de bord).

Pour atteindre cet objectif, on va considérer un déplacement plus général.

Déplacement plus général :

On souhaite écrire une fonction dpg ayant pour arguments a,b,c (listes représentant des tours de Hanoï p_0,p_1,p_2) et $n0$,$n1$,$n2$ (entiers naturels tels que les piles $p0$,$p1$,$p2$ possèdent au moins $n0$,$n1$,$n2$ assiettes et, en considérant les $n0$ assiettes les plus hautes de p_0, les $n1$ assiettes les plus hautes de p_1 et les $n2$ assiettes les plus hautes de p_2, les assiettes considérées sont plus larges que les autres) et déplaçant les $n0$ assiettes les plus hautes de p_0, les $n1$ assiettes les plus hautes de p_1 et les $n2$ assiettes les plus hautes de p_2 sur la pile p_2 (effet de bord)

Question 5. *Ecrire une fonction nommée largeur ayant pour arguments a (liste représentant une tour de Hanoï p) et n (entier naturel non nul tel que p possède au moins n assiettes) et retournant la largeur de l'assiette la plus basse parmi les n assiettes les plus hautes de p. Faire afficher largeur$([15, 12, 9, 7, 6, 3, 2], 3)$.*

Question 6. *Ecrire une fonction nommée "argmax" ayant pour arguments a,b,c (listes représentant*

des tours de Hanoï p_0,p_1,p_2) et $n0$,$n1$,$n2$ (entiers naturels non tous nuls tels que les piles p_0,p_1,p_2 possèdent au moins $n0$,$n1$,$n2$ assiettes) retournant l'indice j défini de la manière suivante :

Si $n0 \neq 0$, on considère $largeur(a, n0)$. Si $n1 \neq 0$, on considère $largeur(b, n1)$. Si $n2 \neq 0$, on considère $largeur(c, n2)$. Parmi les largeurs considérées, on recherche la largeur maximum M et on pose $j = 0$ si $M = largeur(a, n0)$, $j = 1$ si $M = largeur(b, n1)$ et $j = 2$ si $M = largeur(c, n2)$.

On complètera :

```
def argmax(a,b,c,n0,n1,n2) :
    if n0 !=0 :
        M=...
        j=...
        if n1 !=0 and ... :
            M=...
            j=...
        if n2 !=0 and ... :
            M=...
            j=...
    elif n1 !=0 :
        M=...
        j=...
        if n2 !=0 and ... :
            M=
            j=
    else :
        M=
        j=...
    return(j)
```

Définition récursive du déplacement plus général :

Pour effectuer un déplacement plus général vers une pile cible, si il y a des assiettes à déplacer :
Premier cas : l'assiette à déplacer la plus large est déjà sur la pile cible.
On déplace les assiettes à déplacer sauf la plus large sur le pile cible.
Second cas : l'assiette à déplacer la plus large n'est pas sur la pile cible.
On appelle pile intermédaire la pile qui n'est
ni la pile contenant l'assiette à déplacer la plus large, ni la pile cible.
-on commence par déplacer toutes les assiettes à déplacer sauf la plus large sur la pile intermédaire
-puis on déplace l'assiette à déplacer la plus large sur la pile cible
-puis on déplace toutes les assiettes à déplacer sauf la plus large, de la pile intermédiaire vers la pile cible.

Question 7. *Ecrire la fonction "dpg" de manière récursive. La fonction "dpg" devra faire appel aux fonctions "argmax", "de" et "dmp". Puis écrire la fonction "dg". Affecter aux variables k,l,m les listes $[6, 3, 0]$,$[4, 2]$,$[5, 1]$ qui représentent des tours de Hanoï. Affecter à la variable tours la valeur $[k, l, m]$. Faire afficher $tours$. Afin de visualiser, ajouter l'instruction $print(tours)$ dans le corps de la fonction "dpg" juste après chaque instruction appellant la fonction "de". Faire éxécuter $dg(k, l, m)$.*