CM2 : AP

Récursivité.

* Tours de Hanoï

Comment le résoudre ?

-> il faudra déplacer le grand disque de A vers C

Soit n le nombre de disques.

On devra déplacer les n-1 disques de A vers B.
On déplace le grand disque de A vers C
On déplace les n-1 disques de B vers C.

Donc, si je sais résoudre les tours de Hanoï à n-1 disques, je sais résoudre à n disques.

```
def Resoud_hanoi(n:int, depart:str, arrivee):
    Resoud_hanoi(n-1, depart, autre)
    deplace_disque(depart, arrivee)
    Resoud_hanoi(n-1, autre, arrivee)
```

} pseudocode

* Calcul d'une somme fournie par un str

"12+42+31" -> 85.

def evalue_chaine(chaine : str) -> int :

- Si chaine contient au moins un plus, on coupe en deux
- Sinon, c'est facile.

Définition : La résolution d'un problème P sur une donnée a est dit récursif, si parmi toutes les opérations utilisées pour le résoudre on trouve la résolution de P sur b.

* Trace d'un algorithme récursif

Tracer un algo récursif c'est donner à partir des paramètres d'appel la suite des appels récursifs, ainsi que leur valeur de retour.

somme_rec([21, 3, 7]) = 21 + somme_rec([3, 7]) 31 ←
↳ 3 + somme_rec([7]) 10 ←
↳ 7 + somme_rec([]) = 7

* Correction, terminaison

-> terminaison : l'algorithme s'arrête-il ?
-> correction : l'algorithme correspond à sa spécification.

```
def Resoud_hanoi(n:int, depart: str, arrivee):
    if n>0:
        Resoud_hanoi(n-1, depart, autre)
        deplace_disque(depart, arrivee)
        Resoud_hanoi(n-1, autre, arrivee)
```

} version correcte.

Condit° d'arrêt!

```
def est_pair(n:int)->bool:
    if n==0:
        Return True
    else:
        return not est_impair(n)
```

```
def est_impair(n:int)-> bool:
    if n==0:
        Return False
    else:
        return not est_pair(n)
```

paramètre se rapprochent du cas de base.

```
def est_pair(n:int)->bool:
    if n==0:
        Return True
    else:
        return est_pair(n-2)
```

} si n impair, ne termine pas.

* Variant.

soit $P_x = \{x_1; x_2; \dots; x_p\}$ les paramètres de l'algo A
$P_y = \{y_1, y_2 \dots y_p\}$

On suppose que pour résoudre $A(P_x)$, il faut résoudre $A(P_y)$

le variant est un nombre $v(P_x)$ tel que:

-> $v(P_x) \in \mathbb{N}$ ≥ 0
-> $v(P_y) < v(P_x)$

```
def evalue_chaine(chaine : str) -> int :
    i = indice(chaine, '+')
    if i >= 0:
        int(chaine[:i]) + evalue_chaine(chaine[i+1:])
    else:
        return int(chaine)
```

Pour prouver que calcul_somme s'arrête, je dois fournir un variant.

evalue_chaine("21+42+7") = 21 + evalue_chaine("42+7") 21+42+7 = 70.
↳ 42 + evalue_chaine("7") = 42+7

len(chaine) est un entier positif.

et len(chaine[i+1:]) < len(chaine).

Donc len(chaine) est un variant, donc l'algo s'arrête.

pk ?

C'est une suite de nombres entiers strictement décroissante. On ne peut dans ce cas en avoir qu'un nombre fini.

* Récursivité terminale

Exemples :

-> calcul_somme

-> est_pair

```
def est_pair(n : int) -> bool :
    if n == 0:
        return True
    elif n == 1:
        return False
    else:
        return est_pair(n-2)
```

est_pair est récursif terminal car la valeur renvoyée est directement la valeur des appels récursifs.

Un algorithme est dit récursif terminal si les cas récursifs renvoient la valeur de l'appel récursif. (sans changement de valeur)