INF1 : Algorithmique et Programmation

Cours 10 : Récursivité

Domitile Lourdeaux

Université de technologie de Compiègne

Printemps 2024

Voir autrement (1)

De la technologie à la la Science

- **Usages :**
  - Utilisation de logiciels
  - Développement de logiciels
- **Science :**
  - Elaboration de connaissances universelles
  - Raisonnement logique
  - Observations et expérimentations
- **L'informatique théorique** : fondements logiques et mathématiques (Théories de la calculabilité, de la complexité, des automates, etc.)

Sources :

- `https://fr.wikipedia.org/wiki/Machine_de_Turing_universelle`
- `https://slideplayer.fr/slide/11396125/`

Voir autrement (2)

La calculabilité

Branche des mathématiques qui s'intéresse à plusieurs questions fondamentales : qu'est-ce qu'un algorithme ? Peut-on le définir mathématiquement ? Tout problème peut-il être résolu par un algorithme ? Peut-on démontrer mathématiquement qu'un problème n'a pas de solution par un algorithme ? Cette branche est capitale, car le fonctionnement des ordinateurs est fondé sur l'algorithmique ; la calculabilité renseigne donc sur les problèmes que ces machines peuvent résoudre.

Source : `https://www.pourlascience.fr/sd/mathematiques/aux-origines-de-la-calculabilite-3494.php`

Voir autrement (3)

L'indécidabilité est un concept fondamental en logique. Il a plusieurs sens. Au sens de la démonstration, il signifie qu'on ne peut démontrer ni qu'une proposition est vraie ni qu'elle est fausse dans un système d'axiomes ; ici, le contenu sémantique ne permet d'affirmer ni que l'objet est une pipe ni qu'il n'en est pas une. Au sens de la calculabilité, il signifie qu'il n'existe aucun algorithme capable de résoudre un problème (ici déterminer si l'objet est une pipe ou non).

© Christie's Images/Corbis

Source : `https://www.pourlascience.fr/sd/mathematiques/aux-origines-de-la-calculabilite-3494.php`

Vidéo pour comprendre : `https://www.youtube.com/watch?v=13O1qhX4Bqo`

Pour aller plus loin

- TC : **NF93** Fondements scientifiques de l'informatique
- GI : **IA02** Résolution de problèmes et programmation logique

Voir autrement (4)

Enigme simplifiée de Einstein et IA02

Exercice 6 ☆☆☆

les filles du roi Un roi avait trois filles, dont deux étaient brunes et deux avaient les yeux bleus.

1. Modéliser cet énoncé en logique propositionnelle. Montrer que seules 6 variables sont nécessaires.
2. Montrer en utilisant le principe de résolution qu'au moins une des filles du roi était brune aux yeux bleus.

Exercice 7 ☆☆☆ (facultatif)

On souhaite trouver une solution via la logique propositionnelle au problème suivant. Dans une rue sont alignées 3 maisons, numérotées de gauche à droite de 1 à 3. Dans chaque maison habite une unique personne. On veut connaître la couleur de chaque maison et la nationalité de chacun des habitants.

- *Règle 1 :* Chaque maison possède une couleur différente (bleu, vert ou rouge).
- *Règle 2 :* Chaque habitant possède une nationalité différente (Italien, Norvégien ou Espagnol).

- *Indice 1 :* L'Espagnol habite la maison directement à droite de la maison rouge.
- *Indice 2 :* Le Norvégien vit dans la maison bleue.
- *Indice 3 :* L'Italien habite dans la maison n°2.

Questions

1. Coder les règles sous forme de clauses.
2. Donner la formulation logique (sous forme de clauses) des 3 indices.
3. Montrer que le Norvégien habite dans la maison 1. Donner l'arbre de réfutation le démontrant.
4. Quelle est la couleur de la maison 2 ? Prouver-le.

Source : `https://www.irif.fr/~eleph/Autre/maisons.pdf`

(1) *Le norvégien habite la maison n° 1 :*
- ⋆1 $Nor(1)$ — Ax.9

(2) *On boit du lait dans la maison du milieu (n° 3) :*
- ⋆2 $Lait(3)$ — Ax.7

(3) *La maison bleue est la maison n° 2 :*
- 3.1 $Nor(1) \rightarrow Bleu(0) \vee Bleu(2)$ — Ax.13$[i := 1]$
- 3.2 $Bleu(0) \vee Bleu(2)$ — Mod. Pon.(3.1)(1)
- ⋆3 $Bleu(2)$ — Cut(3.2)($N0$)

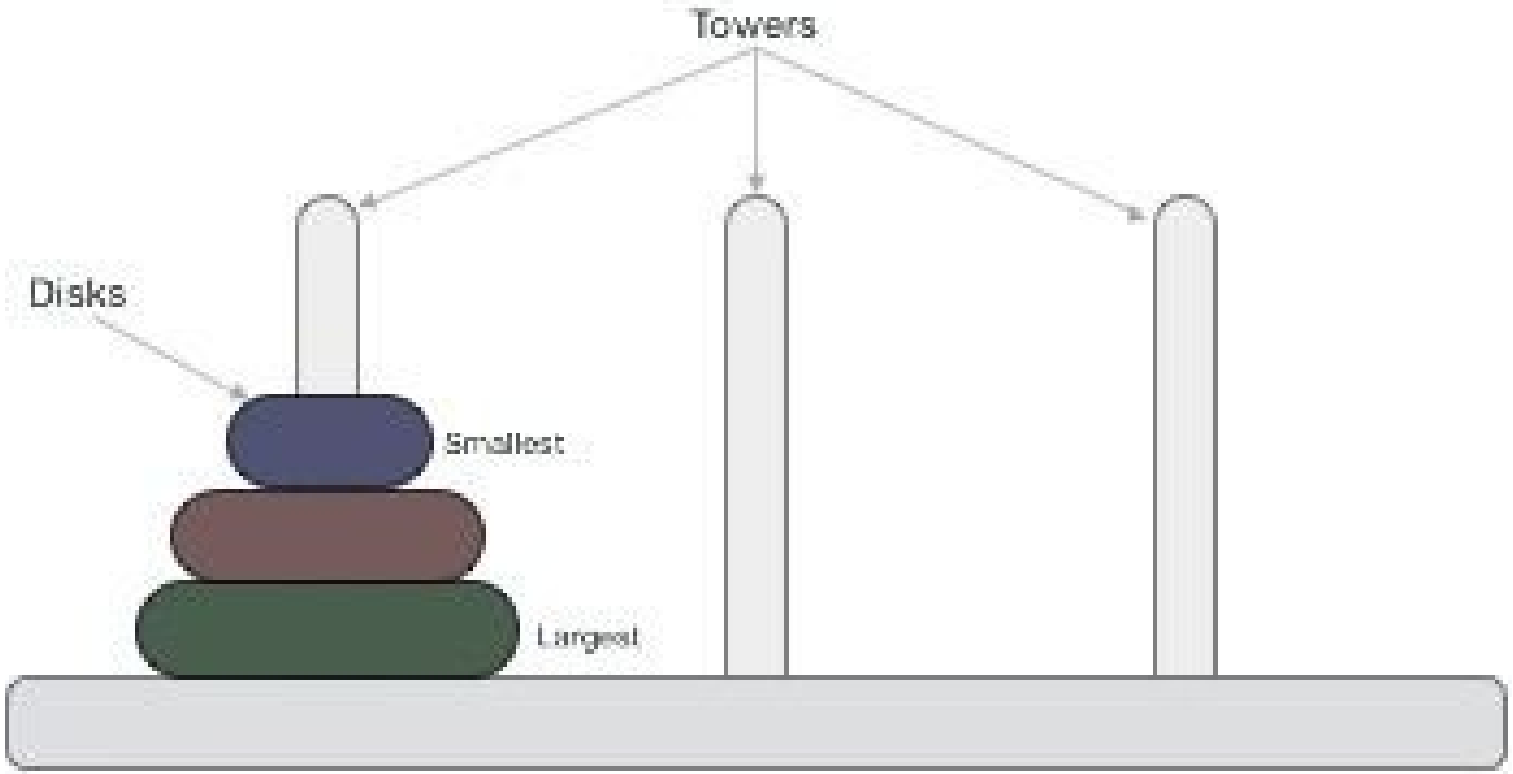
(4) *La maison verte est la maison n° 4 :*
- 4.1 $Vert(1) \rightarrow Blanc(2)$ — Ax.4$[i := 1]$
- 4.2 $\neg Blanc(2)$ — Opp(3)
- 4.3 $\neg Vert(1)$ — Mod. Tol.(4.1)(4.2)
- 4.4 $\neg Vert(2)$ — Opp(3)
- 4.5 $Vert(3) \rightarrow Cafe(3)$ — Ax.5$[i := 3]$
- 4.6 $\neg Cafe(3)$ — Opp(2)
- 4.7 $\neg Vert(3)$ — Mod. Tol.(4.5)(4.6)
- 4.8 $Vert(5) \rightarrow Blanc(6)$ — Ax.4$[i := 5]$
- 4.9 $\neg Vert(5)$ — Mod. Tol.(4.8)($N6$)
- ⋆4 $Vert(4)$ — Cut($\mathcal{I}_{Vert}$)(4.3, 4.4, 4.7, 4.9)

(5) *La maison blanche est la maison n° 5 :*
- 5.1 $Vert(4) \rightarrow Blanc(5)$ — Ax.4$[i := 4]$
- ⋆5 $Blanc(5)$ — Mod. Pon.(5.1)(4)

(6) *La maison rouge est la maison n° 3 :*
- 6.1 $Rouge(1) \rightarrow Ang(1)$ — Ax.1$[i := 1]$
- 6.2 $\neg Ang(1)$ — Opp(1)
- 6.3 $\neg Rouge(1)$ — Mod. Tol.(6.1)(6.2)
- 6.4 $\neg Rouge(2)$ — Opp(3)
- 6.5 $\neg Rouge(4)$ — Opp(4)
- 6.6 $\neg Rouge(5)$ — Opp(5)
- ⋆6 $Rouge(3)$ — Cut($\mathcal{I}_{Rouge}$)(6.3, 6.4, 6.5, 6.6)

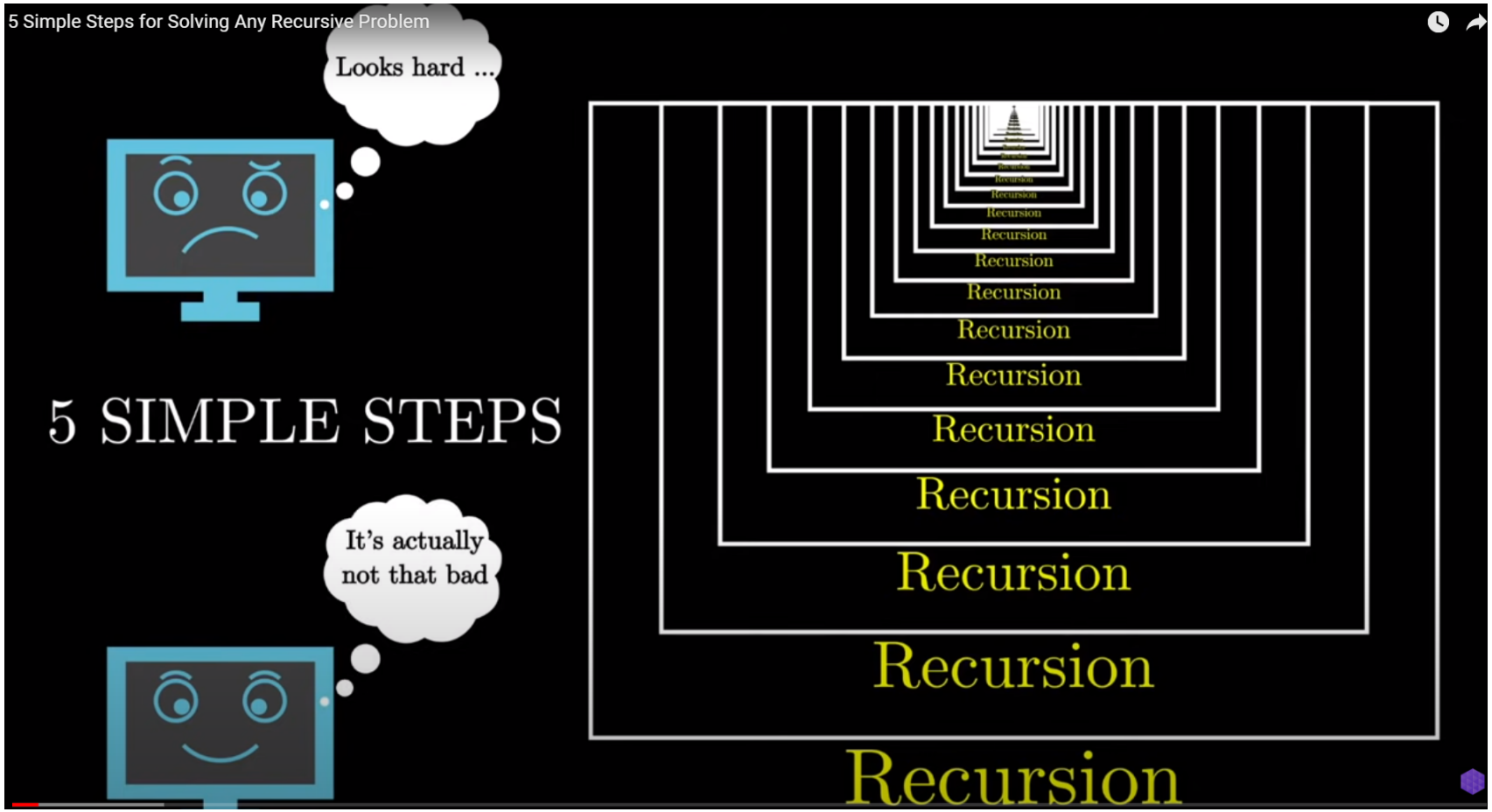
Voir autrement (5)

L'informatique comme un jeu... notion de récursivité



Source : https://fr.quora.com/Quest-ce-que-la-rÃ©cursivitÃ©

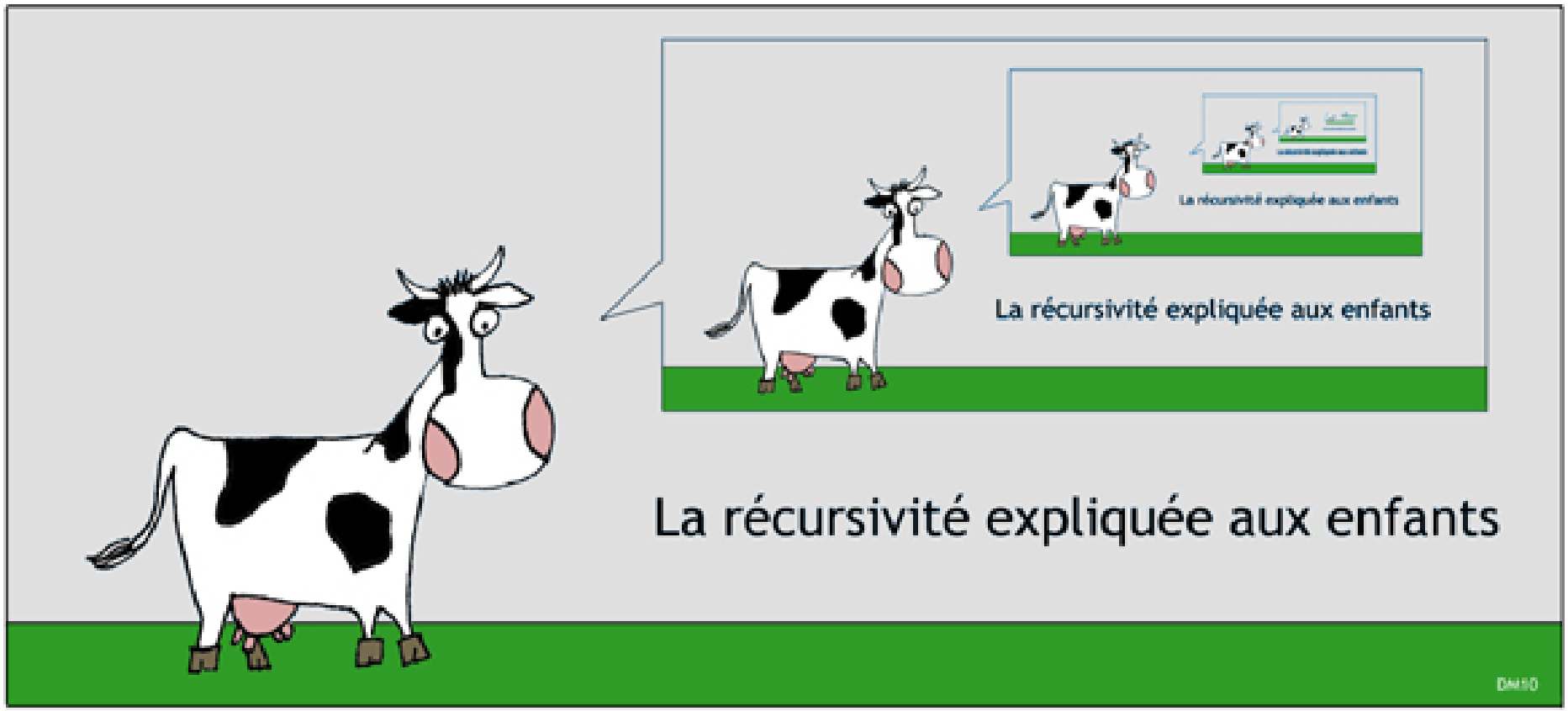
La récursivité : Complexe ?



5 Simple Steps for Solving Any Recursive Problem : `https://youtu.be/ngCos392W4w`

Récursivité expliquée aux enfants

La vache

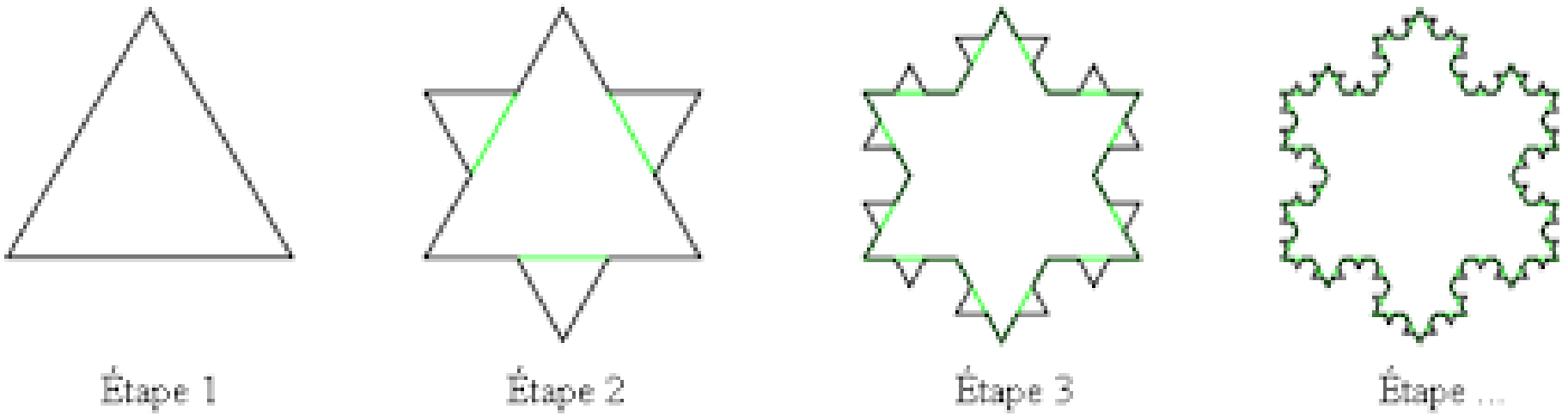


https://www.lecluse.fr/nsi/NSI_T/langages/recursivite/

Le monde est rempli de fractals

Flocon de Koch

Figure fractale - Suite géométrique



La récursivité en Python : `https://youtu.be/LYWxvnW_-JM`

Définition Larousse

récursivité

nom féminin

(de récursif)

1. Propriété que possède une règle ou un élément constituant de pouvoir se répéter de manière théoriquement indéfinie. (C'est une propriété essentielle des règles d'une grammaire générative, celle qui permet d'engendrer un nombre infini de phrases.)

2. Qualité d'un programme informatique récursif.

3. Théorie destinée à fournir un cadre rigoureux à l'étude des notions intuitives de calculabilité et de décidabilité effectives. (Church a montré [1936] que la récursivité est l'équivalent mathématique de la calculabilité effective : la fonction récursive est une fonction rigoureusement calculable.)

https://www.larousse.fr/dictionnaires/francais/rÃ©cursivitÃ©/67271

Définition Informatique

Moyen simple et élégant de résoudre certains problèmes

Définition

Une fonction ou une procédure est dite **récursive** s'il est fait appel à cette fonction ou à cette procédure dans le corps d'instructions qui la définit.

i.e. la fonction (ou la procédure) s'appelle elle-même

Remarque

La récursivité est un moyen de répéter des blocs d'instructions sans utiliser de boucle while ou for

Suite arithmétique

Somme des n premiers entiers non nuls

Soit (u_n) la suite arithmétique définie sur N de raison $r = 1$ et telle que $u_0 = 1$

Calculer $u_0 + u_1 + \ldots + u_5$

Solution mathématique :

$$S = 1 + \ldots + n = \frac{n \times (n+1)}{2}$$

$$S = 1 + 2 + 3 + 4 + 5 = \frac{5 \times (5+1)}{2}$$

Itératif vs Récursif (1)

Somme des n premiers entiers non nuls

Algorithm 1 Somme des n premiers entier : version itérative

```
function somme (n : entier) : entier
    Variables : s, i : entiers
1   début
2       s = 1
         pour i i allant de 2 à n − 1 (exclu) faire
3           s = s + i
4       fin

        retourner s
5   fin
```

Itératif vs Récursif (2)

Somme des n premiers entiers non nuls

- **De manière itérative (répétition) :**
  - Initialisation : $s = 0$
  - $s = 1 + ... + n$
  - On répète plusieurs (n) fois une opération
- **De manière récursive (récursion) :**
  - Initialisation : $f(x, 1) = 1$
  - Relation pour $n > 1$: $f(x, n) = x + f(x, n - 1)$
  - *Ici* $x = 0$

Remarque

La récursivité s'apparente ici à la récurrence en mathématiques

Méthode en Python

Méthode

Lorsque l'on écrit une fonction récursive en Python, on peut partager son code en deux parties :

1. Une **condition d'arrêt** pour stopper les appels récursifs
2. Les **appels récursifs**

Itératif vs Récursif (3)

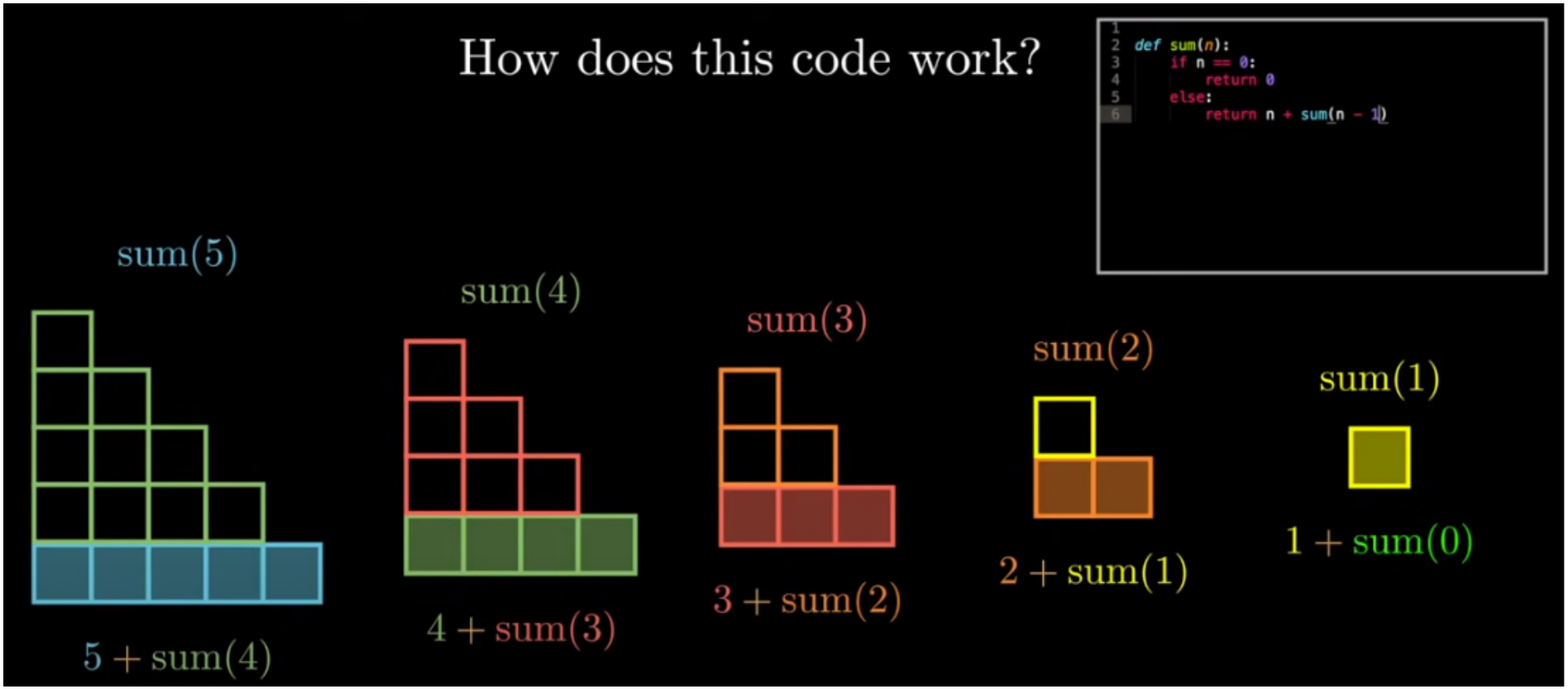
Somme des n premiers entiers non nuls

Algorithm 2 Somme des n premiers entier : version récursive

```
function somme (n : entier) : entier
    début
        si n == 1 alors
            retourner 1
        sinon
            retourner n + somme(n − 1)
        fin
    fin
```

Pour comprendre : principe

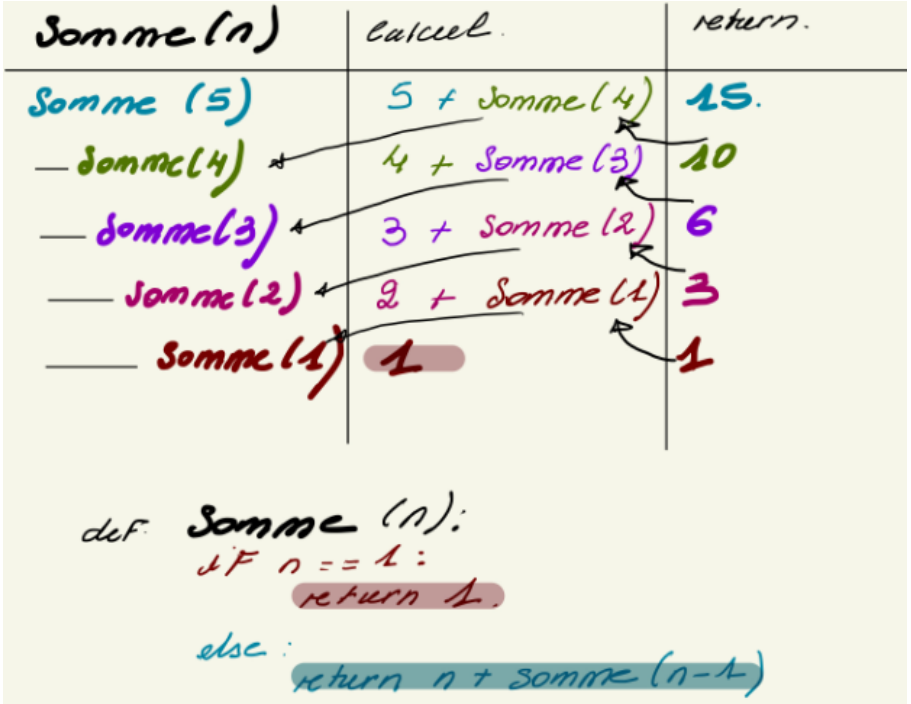
Somme des n premiers entiers non nuls

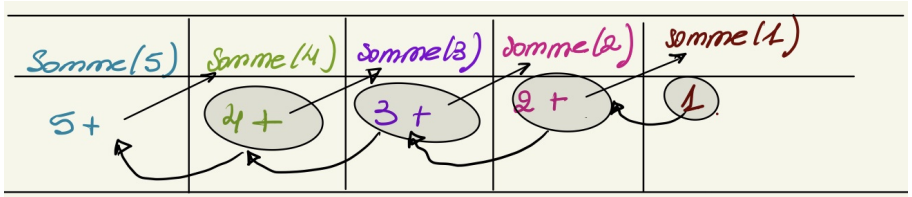


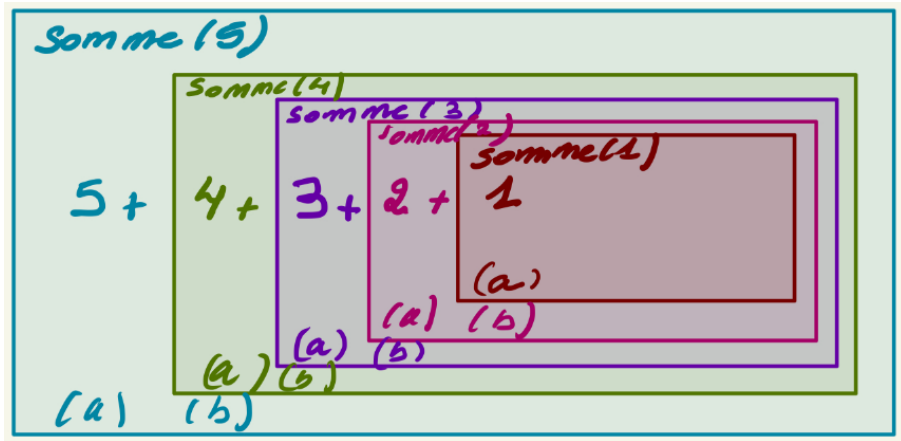
5 Simple Steps for Solving Any Recursive Problem : `https://youtu.be/ngCos392W4w`

Pour comprendre : appels récursifs

Somme des n premiers entiers non nuls







Complexité

Définitions

Le calcul de la complexité d'un algorithme permet de mesurer sa performance

- **Complexité spatiale** : quantité d'espace mémoire nécessaire
- **Complexité temporelle** : nombre d'instructions utilisées

Plusieurs types d'analyses de complexité :

- meilleur des cas
- pire des cas
- en moyenne

Ordres de grandeur : $\mathcal{O}(1), \mathcal{O}(n), \mathcal{O}(n^2), \mathcal{O}(\log(n))$, etc.

Pour en savoir plus

- Voir cours complexité des algorithmes : NF16

Elégance vs complexité (1)

Complexité spatiale

Une fonction récursive peut rapidement être **gourmande en mémoire**

- Message d'erreur :
  - RuntimeError : maximum recursion depth exceeded
- Taille de la **pile d'exécution limitée** (par défaut 1000)
- Pile d'exécution = emplacement mémoire destiné à stocker les paramètres, les variables locales, etc.
- **Pour connaître la limite :**

```
import sys
sys.getrecursionlimit()
```

Elégance vs complexité (2)

Complexité temporelle

Comparer l'efficacité d'algorithmes résolvant le même problème et établir le plus optimal

- Calcule le nombre d'opérations élémentaires (affectation, calcul arithmétique ou logique, etc.)
- Complexité pratique
- Complexité théorique sur une machine idéalisée, indépendant :
  - du langage de programmation
  - du processeur
  - de l'éventuel compilateur employé

Source : `https://cahier-de-prepa.fr/psi-michelet/download?id=239`

Elégance vs complexité (3)

Calcul de factoriel

```
def factorielleRec(n):
    "Factorielle (version récursive) de n, pour n >= 0"
    if n > 1:
        return n * factorielleRec(n-1)
    else:
        return 1
def factorielleIt(n):
    "Factorielle (version itérative) de n, pour n >= 0"
    fact = 1
    for i in range (2, n + 1):
        fact = fact * i
    return fact
```

Elégance vs complexité (4)

Calcul de factoriel

```
def temps(f, n):
    from time import time
    t0 = time()
    f(n)
    t1 = time()
    print(f"Temps d'exécution de {f.__name__} : {round(1E6 * (t1 - t0), 5)} microsecondes")

n = int(input("n = "))
temps(factorielleIt, 9)
temps(factorielleRec, 9)
```

Elégance vs complexité (5)

Calcul de factoriel : résultats

n	15	100	900
récursive	19,00	80,61	1196,62
itérative	13,59	21,71	288,40
math.factorial	1,19	5,00	48.16

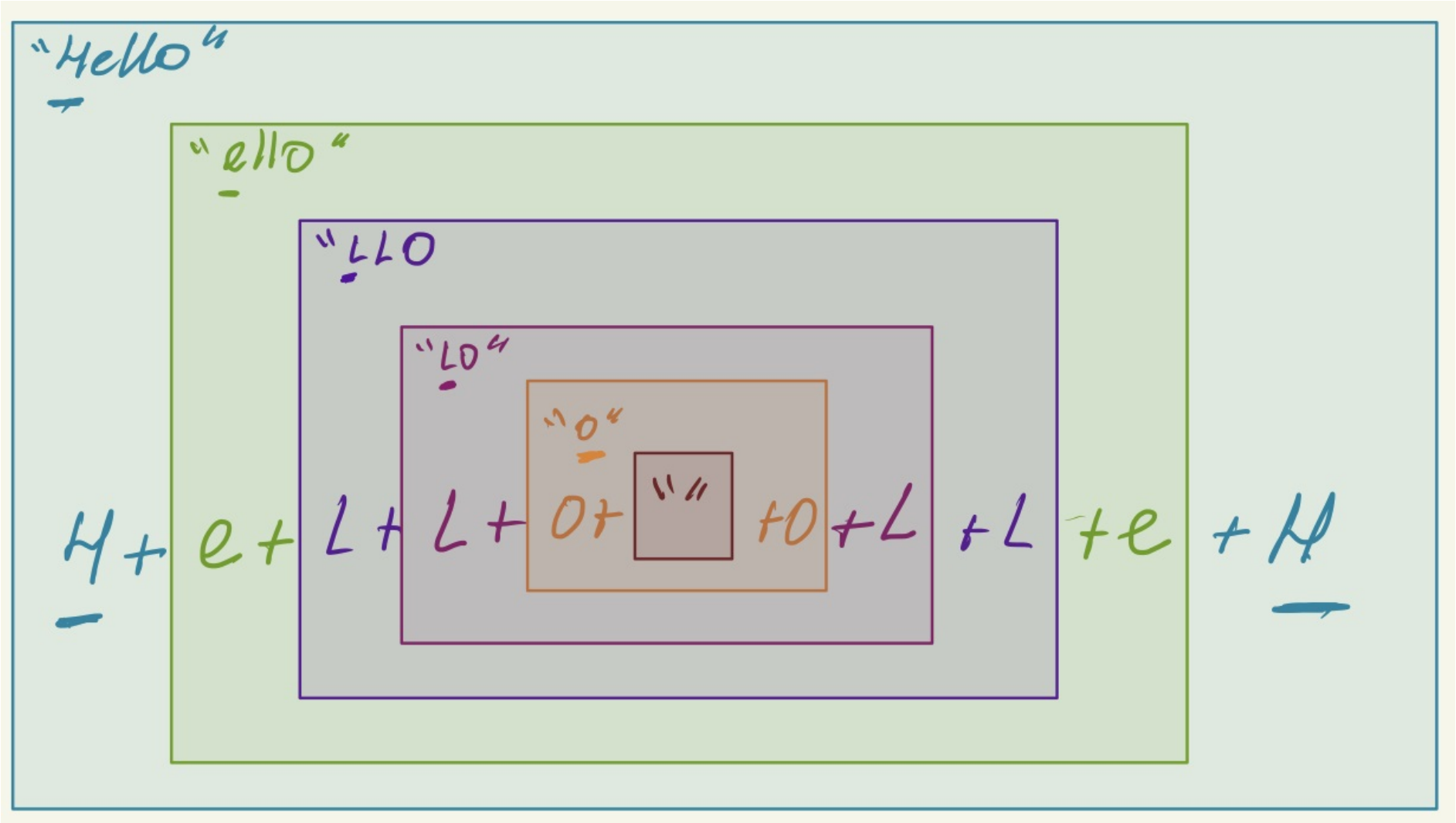
Complexité pratique

- Valeurs indicatives en microsecondes (dépendant ici de la machine)

Construire un palindromme qui n'existe pas (1)

```
def construitPalindrome(ch):
    if ch  :
        return ch[0] + construitPalindrome(ch[1:]) + ch[0]
    else :
        return ""

print(construitPalindrome("Hello"))
```

Construire un palindromme qui n'existe pas (2)

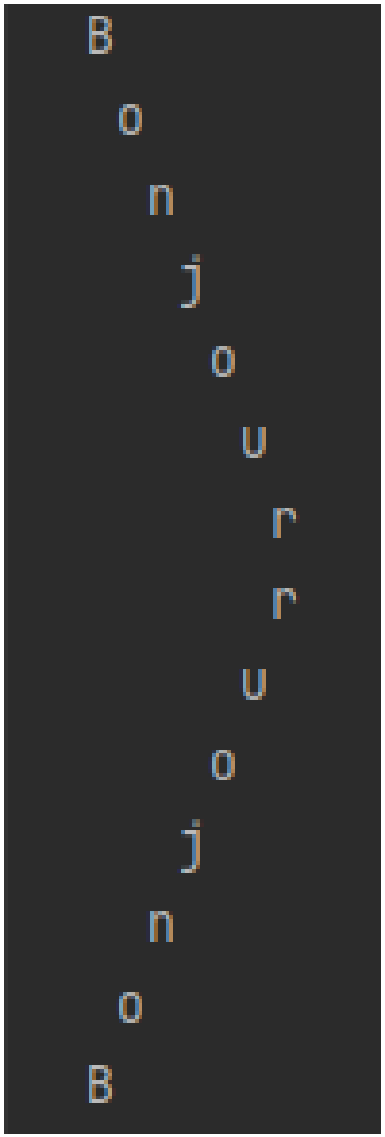


Bonjour (1)

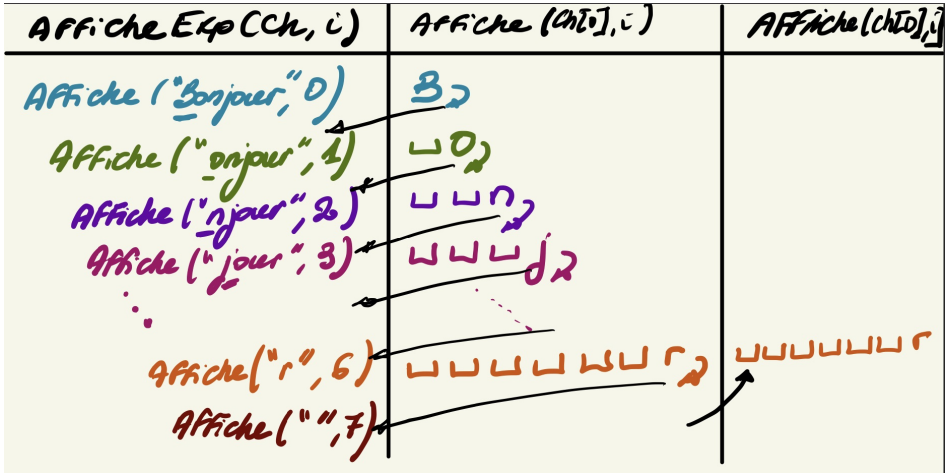
Qu'affiche ce programme si l'utilisateur saisit : "bonjour" ?

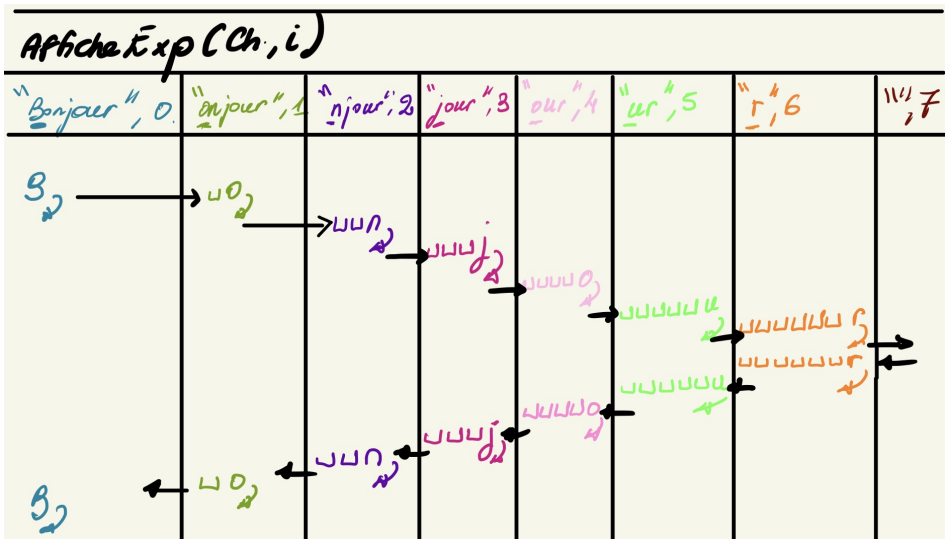
```
def affiche (car, i):
    "Affiche car avec i espaces avant"
    print(" "*i, car)

def afficheExp(ch, i):
    if ch != "":
        affiche(ch[0], i)
        afficheExp(ch[1:], i+1)
        affiche(ch[0], i)

chaine = input("Saisir une chaine de caract res : ")
afficheExp(chaine, 0)
```

Bonjour (2)



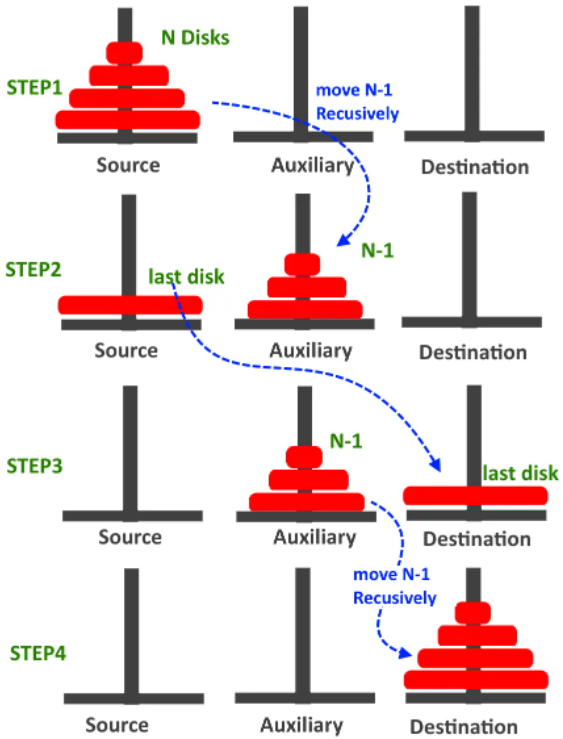
Bonjour (3)


AfficheExp(Ch, i)
Affiche(ch[i], i)
Affiche(ch[i], i)
Affiche("Bonjour", 0)
Affiche("onjour", 1)
Affiche("njour", 2)
Affiche("jour", 3)
Affiche("r", 6)
Affiche("", 7)


AfficheExp(Ch, i)
"Bonjour", 0
"onjour", 1
"njour", 2
"jour", 3
"our", 4
"ur", 5
"r", 6
"", 7

Tour de Hanoï (1)

But : Déplacer la pile de la tour 1 à la tour 3, en ne déplaçant qu'un disque à la fois, et en s'assurant qu'aucun disque ne repose sur un disque de plus petite dimension



Source : `https://lycee-benoit.tech/NSI/term/recursivite/`

Tour de Hanoï (2)

Principe

- Déplacer la pile des n-1 premiers disques de la tour 1 à la tour 2
- Déplacer le dernier disque de la tour 1 à la tour 3
- Déplacer les n-1 disques de la tour 2 à la tour 3

Condition d'arrêt : nb de disques = 0 (il n'y a plus de disque à déplacer)

Tour de Hanoï (3)

Programme et exécution

```
def hanoi(nbDisques, t1, t3, t2):
    if nbDisques > 0 :
        hanoi(nbDisques - 1, t1, t2, t3)
        print('Déplacer disque de ', t1,' à ', t3)
        hanoi(nbDisques - 1, t2, t3, t1)
```

Arrêt quand
il n'y a plus de disque

Test :

hanoi(4,1,3,2)

```
Déplacer disque de  1  à  2
Déplacer disque de  1  à  3
Déplacer disque de  2  à  3
Déplacer disque de  1  à  2
Déplacer disque de  3  à  1
Déplacer disque de  3  à  2
Déplacer disque de  1  à  2
Déplacer disque de  1  à  3
Déplacer disque de  2  à  3
Déplacer disque de  2  à  1
Déplacer disque de  3  à  1
Déplacer disque de  2  à  3
Déplacer disque de  1  à  2
Déplacer disque de  1  à  3
Déplacer disque de  2  à  3
```

Questions...