



Fonctions et récursion

Informatique pour tou(te)s - semaines 40 à 42

Guillaume CONNAN

mise à jour 3 octobre 2015

Lycée Clemenceau - MP / MP*

Sommaire

- 1 Fonctions pures
- 2 Fonctions impures
- 3 lambda expressions

- 4 Composition
- 5 Récursion
- 6 Raisonnement par récurrence
- 7 Suites de Fibonacci

Sommaire

1 Fonctions pures

2 Fonctions impures

3 lambda expressions

4 Composition

5 Récursion

6 Raisonnement par récurrence

7 Suites de Fibonacci

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE \times CODOMAINE
mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE \times CODOMAINE
mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE \times CODOMAINE
mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de DOMAINE \times CODOMAINE
mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de $\text{DOMAINE} \otimes \text{CODOMAINE}$
mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de $\text{DOMAINE} \otimes \text{CODOMAINE}$

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de $\text{DOMAINE} \otimes \text{CODOMAINE}$

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de $\text{DOMAINE} \times \text{CODOMAINE}$

...mais il n'y a pas deux couples ayant le même premier élément

Départ DOMAINE

Arrivée CODOMAINE

Une fonction est une relation

C'est donc une partie (sous-ensemble) de $\text{DOMAINE} \times \text{CODOMAINE}$

...mais il n'y a pas deux couples ayant le même premier élément

Donc la fonction *double* de domaine $\{1, 2, 3, \dots\}$ est

$$\{(1, 2), (2, 4), (3, 6), \dots\}$$

Donc la fonction *double* de domaine $\{1, 2, 3, \dots\}$ est

$$\{(1, 2), (2, 4), (3, 6), \dots\}$$

La fonction *addition* de domaine $\{1, 2, 3, \dots\} \otimes \{1, 2, 3, \dots\}$ est

$$\{((1, 1), 2), ((1, 2), 3), \dots, ((5, 7), 12), ((5, 8), 13), \dots\}$$

La fonction *addition* de domaine $\{1, 2, 3, \dots\} \otimes \{1, 2, 3, \dots\}$ est

$$\{((1, 1), 2), ((1, 2), 3), \dots, ((5, 7), 12), ((5, 8), 13), \dots\}$$

Mais bon, ça manque un peu de dynamisme.



Départ → Arrivée

Départ → Arrivée

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$.
- La majuscule correspondant a un code diminué de 20.
- La fonction doit retourner une chaîne de longueur 1.

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$.
- La majuscule correspondante a un code diminué de 26.
- La fonction doit retourner une chaîne de longueur 1.

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

Exemple 1

On veut mettre une chaîne de caractères en majuscules

- On commence par créer une fonction qui met une chaîne de longueur 1 en majuscule.
- Cela ne concerne que les lettres minuscules.
- On manipule les unicodes correspondant.
- Les lettres minuscules ont un code entre 97 et $97 + 26$
- La majuscule correspondant a un code diminué de 2^5
- La fonction doit retourner une chaîne de longueur 1

```
In [62]: ?chr
```

```
Docstring:
```

```
chr(i) -> Unicode character
```

```
Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
```

```
Type:      builtin_function_or_method
```

```
In [63]: ?ord
```

```
Docstring:
```

```
ord(c) -> integer
```

```
Return the integer ordinal of a one-character string.
```

```
Type:      builtin_function_or_method
```

```
Prelude Data.Char> :t chr
```

```
chr :: Int -> Char
```

```
Prelude Data.Char> :t ord
```

```
ord :: Char -> Int
```

In [62]: `?chr`

Docstring:

`chr(i)` -> Unicode character

Return a Unicode string of one character with ordinal `i`; `0 <= i <= 0x10ffff`.

Type: `builtin_function_or_method`

In [63]: `?ord`

Docstring:

`ord(c)` -> integer

Return the integer ordinal of a one-character string.

Type: `builtin_function_or_method`

```
Prelude Data.Char> :t chr
```

```
chr :: Int -> Char
```

```
Prelude Data.Char> :t ord
```

```
ord :: Char -> Int
```

```
In [62]: ?chr
```

```
Docstring:
```

```
chr(i) -> Unicode character
```

```
Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
```

```
Type:      builtin_function_or_method
```

```
In [63]: ?ord
```

```
Docstring:
```

```
ord(c) -> integer
```

```
Return the integer ordinal of a one-character string.
```

```
Type:      builtin_function_or_method
```

```
Prelude Data.Char> :t chr
```

```
chr :: Int -> Char
```

```
Prelude Data.Char> :t ord
```

```
ord :: Char -> Int
```

```
def car_en_maj(car) :  
    ascii = ord(car)  
    if 97 <= ascii <= 97 + 26 :  
        return chr(ascii - 32)  
    else :  
        return car
```

```
def car_en_maj(car) :  
    ascii = ord(car)  
    if 97 <= ascii <= 97 + 26 :  
        return chr(ascii - 32)  
    return car
```

```
def car_en_maj(car) :  
    ascii = ord(car)  
    if 97 <= ascii <= 97 + 26 :  
        return chr(ascii - 32)  
    else :  
        return car
```

```
def car_en_maj(car) :  
    ascii = ord(car)  
    if 97 <= ascii <= 97 + 26 :  
        return chr(ascii - 32)  
    return car
```

Sommaire

- 1 Fonctions pures
- 2 Fonctions impures**
- 3 lambda expressions

- 4 Composition
- 5 Récursion
- 6 Raisonnement par récurrence
- 7 Suites de Fibonacci

Print

Et pourquoi pas `print` ?

`ord` prend un argument de type `string`, retourne un argument de type `int` ET NE FAIT QUE ÇA.
C'est une fonction PURE

Print

Et pourquoi pas `print` ?

`ord` prend un argument de type `string`, retourne un argument de type `int` ET NE FAIT QUE ÇA.

C'est une fonction PURE

Print

Et pourquoi pas `print` ?

`ord` prend un argument de type `string`, retourne un argument de type `int` ET NE FAIT QUE ÇA.

C'est une fonction PURE

Print

Et pourquoi pas `print` ?

`ord` prend un argument de type `string`, retourne un argument de type `int` ET NE FAIT QUE ÇA.

C'est une fonction PURE



8

VALID

MORROW, JEROME
011010100-09564

RED GTACATGACTAAGTTAC EYES:BLUE TACCTGTCA
AGCTTGACCTCCCTGAAGTCACCAGTTCGATGCTTGAC
GNQ9.5612 = VALIDITY JE7542DAN

Print

```
In [34]: type(print(2))
```

```
2
```

```
Out[34]: NoneType
```

```
In [35]: print(print(2))
```

```
2
```

```
None
```

```
In [36]: print(print(2),print('GNU'))
```

```
2
```

```
GNU
```

```
None None
```

```
In [37]: [ print(a) for a in range(5) ]
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
Out[37]: [None, None, None, None, None]
```

```
In [38]: b = [ print(a) for a in range(5) ]
```

```
0  
1  
2  
3  
4
```

```
In [39]: b
```

```
Out[39]: [None, None, None, None, None]
```

`print` est IMPURE !

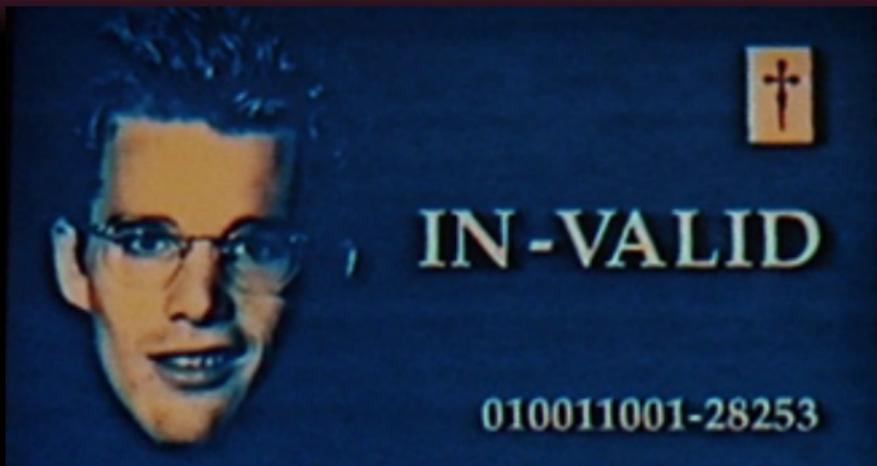
```
In [38]: b = [ print(a) for a in range(5) ]
```

```
0  
1  
2  
3  
4
```

```
In [39]: b
```

```
Out[39]: [None, None, None, None, None]
```

print est IMPURE !



IN-VALID

010011001-28253

TACATGACTAAGTTAC MYOPIA: TACCTGTCATT I
TCCACCATGTACCTACTTCCAAATGCTTGACCAAT

GQ 3.4071 = DEFICIENCY LI
*SUSP. DE·GENE·ERATE

`print` renvoie toujours `None` mais a un EFFET SECONDAIRE (*side effect*) qui affiche quelque chose dans le monde extérieur.

```
def plus_chaine(x) :  
    global a  
    return x + ' ' + a
```

```
In [102]: a = 'Dave'
```

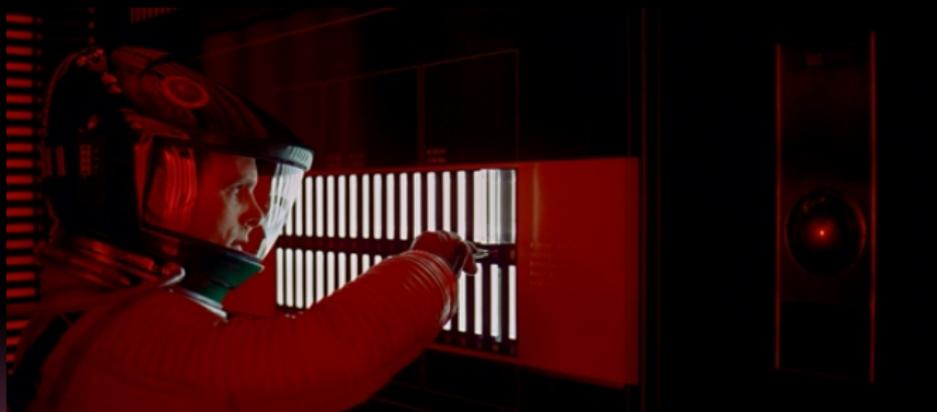
```
In [103]: plus_chaine('Bonjour')
```

```
Out[103]: 'Bonjour Dave'
```

```
In [104]: a = 'HAL'
```

```
In [105]: plus_chaine('Bonjour')
```

```
Out[105]: 'Bonjour HAL'
```



```
In [102]: a = 'Dave'
```

```
In [103]: plus_chaine('Bonjour')
```

```
Out[103]: 'Bonjour Dave'
```

```
In [104]: a = 'HAL'
```

```
In [105]: plus_chaine('Bonjour')
```

```
Out[105]: 'Bonjour HAL'
```





IN-VALID

010011001-28253

TACATGACTAAGTTAC MYOPIA: TACCTGTCATT I
TCCACCATGTACCTACTTCCAAATGCTTGACCAAT

GQ 3.4071 = DEFICIENCY LI
*SUSP. DE·GENE·ERATE

==GO



<http://golang.org>



Rust



mozilla

Comment expliquer :

```
In [1]: L = [1, 2, 3]
```

```
In [2]: type(L)
```

```
Out[2]: list
```

```
In [3]: L = L.append(4)
```

```
In [4]: L
```

```
In [5]: type(L)
```

```
Out[5]: NoneType
```

Comment expliquer :

```
In [6]: L = L.append(5)
```

```
-----  
AttributeError
```

```
Traceback (most recent call last)
```

```
<ipython-input-6-1cadcf93bdc1> in <module>()  
----> 1 L = L.append(5)
```

```
AttributeError: 'NoneType' object has no attribute 'append'
```

Comment expliquer :

```
In [7]: L = [1, 2, 3]
```

```
In [8]: L = L.pop()
```

```
In [9]: L = L + [4]
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-9-fa09307a637a> in <module>()  
----> 1 L = L + [4]
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

Sommaire

- 1 Fonctions pures
- 2 Fonctions impures
- 3 lambda expressions**

- 4 Composition
- 5 Récursion
- 6 Raisonnement par récurrence
- 7 Suites de Fibonacci

Les fonctions sont des expressions comme les autres.

```
def transforme_une_lettre(transformation, lettre) :  
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')  
Out[119]: 'G'
```

```
def cesar(lettre) :  
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')  
Out[120]: 'j'
```

```
def transforme_une_lettre(transformation, lettre) :  
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
```

```
Out[119]: 'G'
```

```
def cesar(lettre) :  
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
```

```
Out[120]: 'j'
```

```
def transforme_une_lettre(transformation, lettre) :  
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
```

```
Out[119]: 'G'
```

```
def cesar(lettre) :  
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
```

```
Out[120]: 'j'
```

```
def transforme_une_lettre(transformation, lettre) :  
    return transformation(lettre)
```

```
In [119]: transforme_une_lettre(car_en_maj, 'g')
```

```
Out[119]: 'G'
```

```
def cesar(lettre) :  
    return chr( ord(lettre) + 3 )
```

```
In [120]: transforme_une_lettre(cesar, 'g')
```

```
Out[120]: 'j'
```

λ -expressions



$$x \mapsto 2x + 1$$

$$\lambda x \cdot 2x + 1$$

```
lambda x : 2*x + 1
"Une fonction qui prend x et retourne 2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
```

```
Out[121]: 11
```

$$x \mapsto 2x + 1$$

$$\lambda x. 2x + 1$$

```
lambda x : 2*x + 1
"Une fonction qui prend x et retourne 2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
```

```
Out[121]: 11
```

$$x \mapsto 2x + 1$$

$$\lambda x. 2x + 1$$

```
lambda      x      :      2*x + 1
"Une fonction qui prend x et retourne 2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
```

```
Out[121]: 11
```

$$x \mapsto 2x + 1$$

$$\lambda x. 2x + 1$$

```
lambda      x      :      2*x + 1
"Une fonction qui prend x et retourne 2*x + 1"
```

```
In [121]: (lambda x : 2*x + 1)(5)
```

```
Out[121]: 11
```

```
In [1]: transforme_une_lettre(lambda c : chr( ord(c) + 5 ), 'a')  
Out[1]: 'f'
```

```
def cesar(decalage) :  
    return lambda lettre : chr( ord(lettre) + decalage )
```

```
In [127]: cesar(3)
```

```
Out[127]: <function __main__.cesar.<locals>.<lambda>>
```

```
In [128]: cesar(3)('a')
```

```
Out[128]: 'd'
```

```
In [129]: cesar(5)('a')
```

```
Out[129]: 'f'
```

```
def cesar(decalage) :  
    return lambda lettre : chr( ord(lettre) + decalage )
```

```
In [127]: cesar(3)
```

```
Out[127]: <function __main__.cesar.<locals>.<lambda>>
```

```
In [128]: cesar(3)('a')
```

```
Out[128]: 'd'
```

```
In [129]: cesar(5)('a')
```

```
Out[129]: 'f'
```

```
def cesar(decalage) :  
    return lambda lettre : chr( ord(lettre) + decalage )
```

```
In [127]: cesar(3)
```

```
Out[127]: <function __main__.cesar.<locals>.<lambda>>
```

```
In [128]: cesar(3>('a'))
```

```
Out[128]: 'd'
```

```
In [129]: cesar(5>('a'))
```

```
Out[129]: 'f'
```

Sommaire

- 1 Fonctions pures
- 2 Fonctions impures
- 3 lambda expressions

- 4 **Composition**
- 5 Récursion
- 6 Raisonnement par récurrence
- 7 Suites de Fibonacci

Composition de fonctions

Défi ISI n° 1 :

Shell

```
echo $@ | tr " " "\n" | nl | sort -nr | cut -f2- | tr "\n" " " | cat ; echo
```

Shell

```
echo $@ | tr " " "\n" | tac | tr "\n" " " | cat ; echo
```

Shell

```
$ ./defi2.sh un deux trois quatre
quatre trois deux un
$
```

Composition de fonctions

Défi ISI n° 1 :

Shell

```
echo $@ | tr " " "\n" | nl | sort -nr | cut -f2- | tr "\n" " " | cat ; echo
```

Shell

```
echo $@ | tr " " "\n" | tac | tr "\n" " " | cat ; echo
```

Shell

```
$ ./defi2.sh un deux trois quatre
quatre trois deux un
$
```

Composition de fonctions

Défi ISI n° 1 :

Shell

```
echo $@ | tr " " "\n" | nl | sort -nr | cut -f2- | tr "\n" " " | cat ; echo
```

Shell

```
echo $@ | tr " " "\n" | tac | tr "\n" " " | cat ; echo
```

Shell

```
$ ./defi2.sh un deux trois quatre
quatre trois deux un
$
```

Dans un vrai langage de programmation :

```
(.) :: (b -> c) -> (a -> b) -> a -> c  
f . g = \x -> f (g x)
```

Défi

Écrire une fonction qui renvoie la longueur du mot le plus long d'une chaîne de caractères donnée en argument.

```
In [1]: long_mot_plus_long("Vos beaux yeux d'amour mourir me font Belle  
      ↪ Marquise")
```

```
Out[1]: 8
```

- Il faudrait une fonction qui éclate la phrase en liste de mots
- Il faudrait une fonction qui renvoie la liste des longueurs de ces mots
- Il faudrait une fonction qui calcule le maximum des éléments d'une liste

- Il faudrait une fonction qui éclate la phrase en liste de mots
- Il faudrait une fonction qui renvoie la liste des longueurs de ces mots
- Il faudrait une fonction qui calcule le maximum des éléments d'une liste

- Il faudrait une fonction qui éclate la phrase en liste de mots
- Il faudrait une fonction qui renvoie la liste des longueurs de ces mots
- Il faudrait une fonction qui calcule le maximum des éléments d'une liste

$\text{String} \xrightarrow{\text{eclate}} [\text{String}] \xrightarrow{\text{longueurs}} [\text{Int}] \xrightarrow{\text{maximum}} \text{Int}$

$s \xrightarrow{\text{eclate}} \text{eclate } s \xrightarrow{\text{longueurs}} \text{longueurs } (\text{eclate } s) \xrightarrow{\text{maximum}} \text{maximum } (\text{longueurs } (\text{eclate } s))$

$\text{String} \xrightarrow{\text{maximum} \circ \text{longueurs} \circ \text{eclate}} \text{Int}$

$\text{String} \xrightarrow{\text{eclate} | \text{longueurs} | \text{maximum}} \text{Int}$

$$\text{String} \xrightarrow{\text{eclate}} [\text{String}] \xrightarrow{\text{longueurs}} [\text{Int}] \xrightarrow{\text{maximum}} \text{Int}$$
$$s \xrightarrow{\text{eclate}} \text{eclate } s \xrightarrow{\text{longueurs}} \text{longueurs (eclate } s) \xrightarrow{\text{maximum}} \text{maximum (longueurs (eclate } s))$$
$$\text{String} \xrightarrow{\text{maximum} \circ \text{longueurs} \circ \text{eclate}} \text{Int}$$
$$\text{String} \xrightarrow{\text{eclate} | \text{longueurs} | \text{maximum}} \text{Int}$$

$$\text{String} \xrightarrow{\text{eclate}} [\text{String}] \xrightarrow{\text{longueurs}} [\text{Int}] \xrightarrow{\text{maximum}} \text{Int}$$
$$s \xrightarrow{\text{eclate}} \text{eclate } s \xrightarrow{\text{longueurs}} \text{longueurs (eclate } s) \xrightarrow{\text{maximum}} \text{maximum (longueurs (eclate } s))$$
$$\text{String} \xrightarrow{\text{maximum} \circ \text{longueurs} \circ \text{eclate}} \text{Int}$$
$$\text{String} \xrightarrow{\text{eclate} | \text{longueurs} | \text{maximum}} \text{Int}$$

$$\text{String} \xrightarrow{\text{eclate}} [\text{String}] \xrightarrow{\text{longueurs}} [\text{Int}] \xrightarrow{\text{maximum}} \text{Int}$$
$$s \xrightarrow{\text{eclate}} \text{eclate } s \xrightarrow{\text{longueurs}} \text{longueurs (eclate } s) \xrightarrow{\text{maximum}} \text{maximum (longueurs (eclate } s))$$
$$\text{String} \xrightarrow{\text{maximum} \circ \text{longueurs} \circ \text{eclate}} \text{Int}$$
$$\text{String} \xrightarrow{\text{eclate} \mid \text{longueurs} \mid \text{maximum}} \text{Int}$$

```
def long_mot_plus_long(cs) :  
    return maximum( longueurs( eclate( cs ) ) )
```

```
In [14]: eclate("Vos beaux yeux")  
Out[14]: ['Vos', 'beaux', 'yeux']
```

```
def eclate(chaine) :  
    return chaine.split()
```

```
In [14]: eclate("Vos beaux yeux")  
Out[14]: ['Vos', 'beaux', 'yeux']
```

```
def eclate(chaine) :  
    return chaine.split()
```

```
In [15]: longueurs(['Vos', 'beaux', 'yeux'])  
Out[15]: [3, 5, 4]
```

```
def longueurs(xs) :  
    return [ len(x) for x in xs ]
```

```
In [15]: longueurs(['Vos', 'beaux', 'yeux'])  
Out[15]: [3, 5, 4]
```

```
def longueurs(xs) :  
    return [ len(x) for x in xs ]
```

```
def maximum(xs) :  
    maxi = xs[0]  
    for elmt in xs[1:] :  
        if elmt > maxi :  
            maxi = elmt  
    return maxi
```

```
def eclate(chaine) :  
    return chaine.split()  
  
def longueurs(xs) :  
    return [ len(x) for x in xs ]  
  
def maximum(xs) :  
    maxi = xs[0]  
    for elmt in xs[1:] :  
        if elmt > maxi :  
            maxi = elmt  
    return maxi  
  
def long_mot_plus_long(cs) :  
    return maximum(longueurs(eclate(cs)))
```

```
def compose2(f,g) :  
    return lambda x: f(g(x))
```

```
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")  
Out[28]: [2, 4, 5]
```

```
long_not_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

```
def compose2(f,g) :  
    return lambda x: f(g(x))
```

```
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")  
Out[28]: [2, 4, 5]
```

```
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

```
def compose2(f,g) :  
    return lambda x: f(g(x))
```

```
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")  
Out[28]: [2, 4, 5]
```

```
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

```
def compose2(f,g) :  
    return lambda x: f(g(x))
```

```
compose2 = lambda f,g : lambda x : f(g(x))
```

```
In [28]: compose2(longueurs,eclate)("un deux trois")  
Out[28]: [2, 4, 5]
```

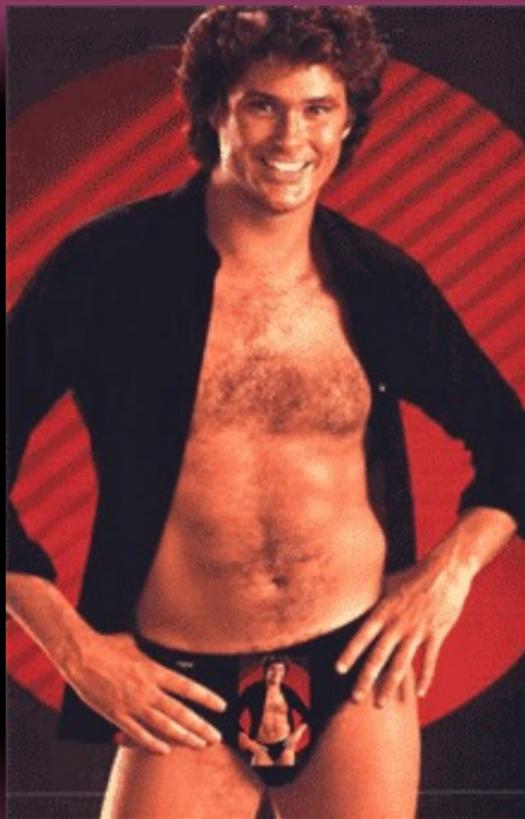
```
long_mot_plus_long = compose2( maximum, compose2(longueurs, eclate) )
```

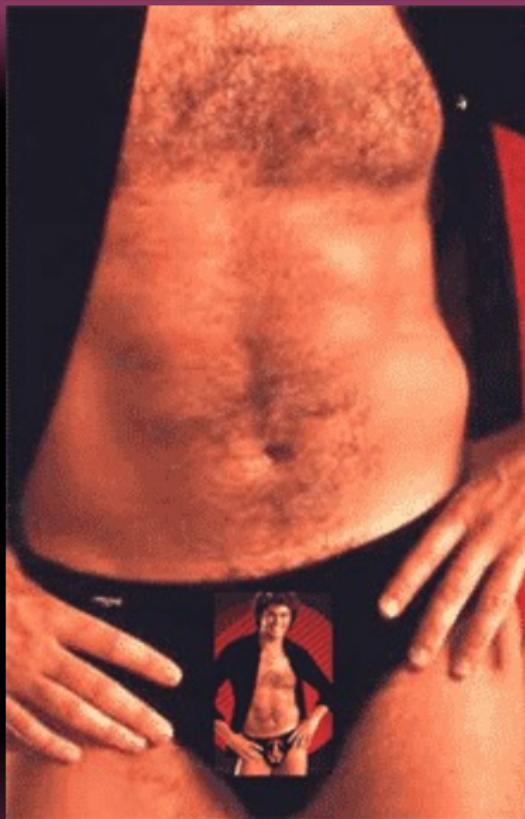
Sommaire

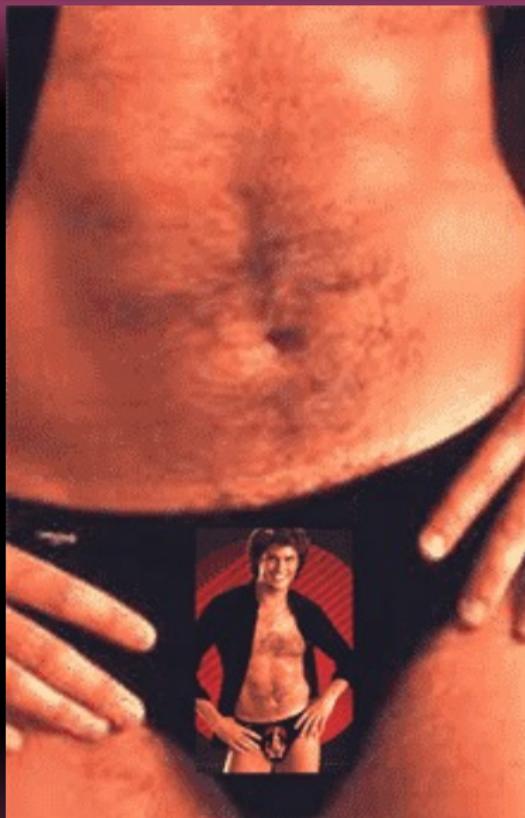
- 1 Fonctions pures
- 2 Fonctions impures
- 3 lambda expressions

- 4 Composition
- 5 **Récursion**
- 6 Raisonnement par récurrence
- 7 Suites de Fibonacci



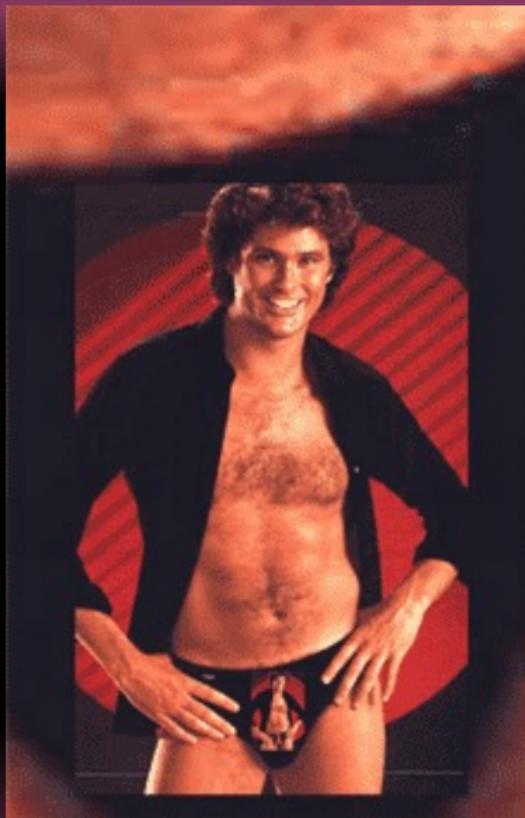


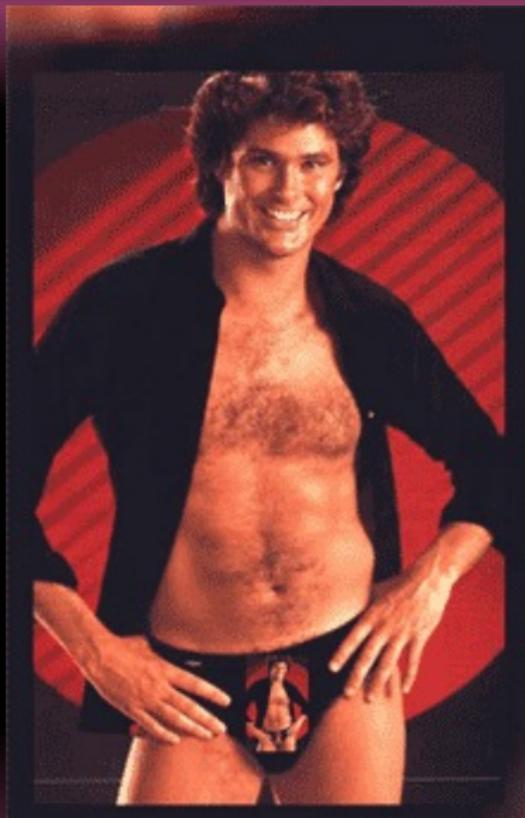


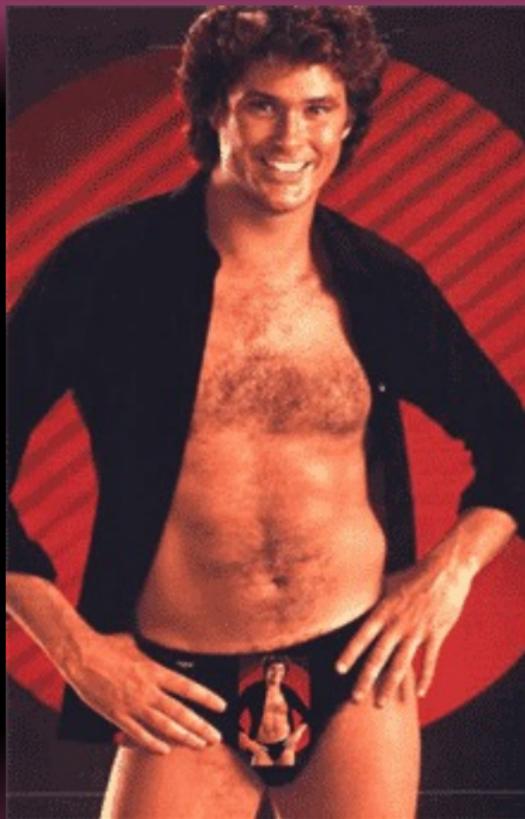




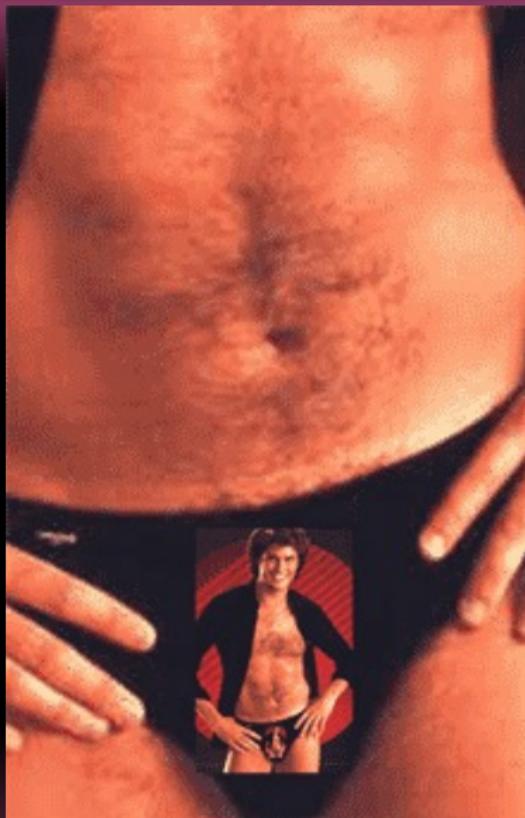






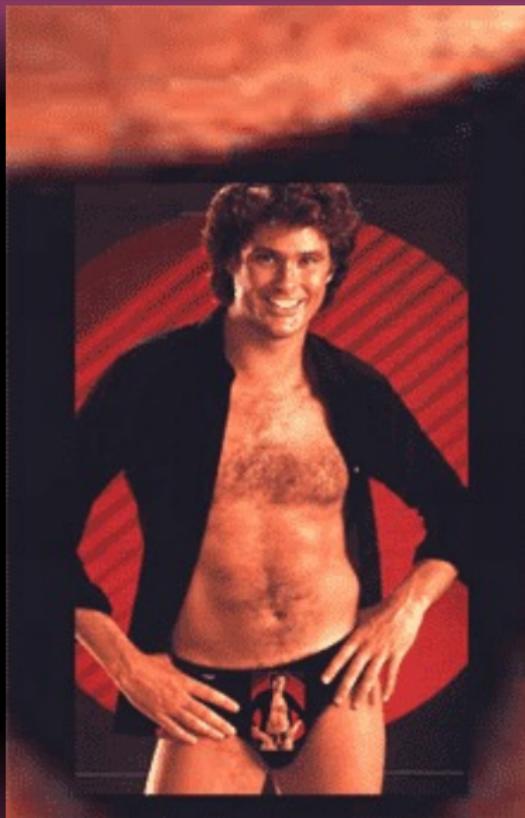


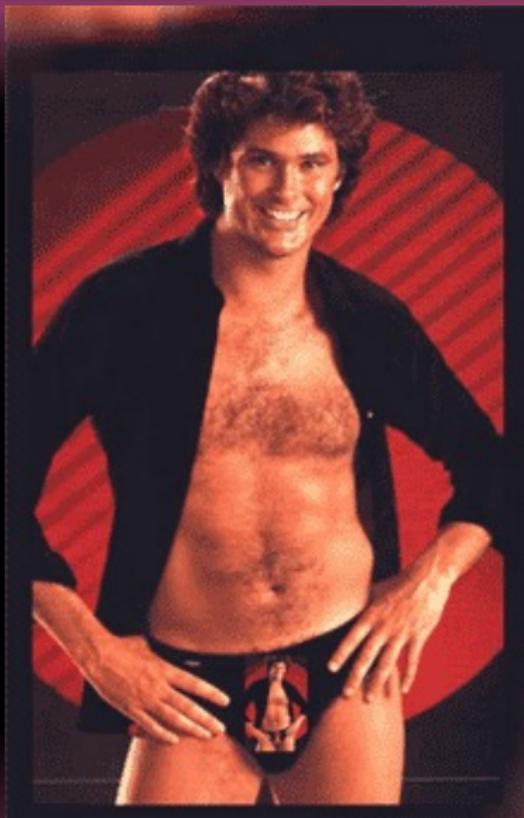






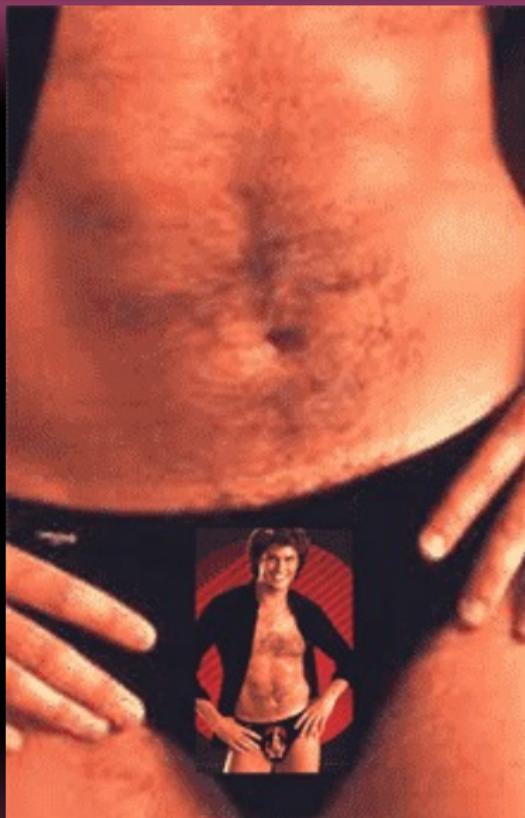






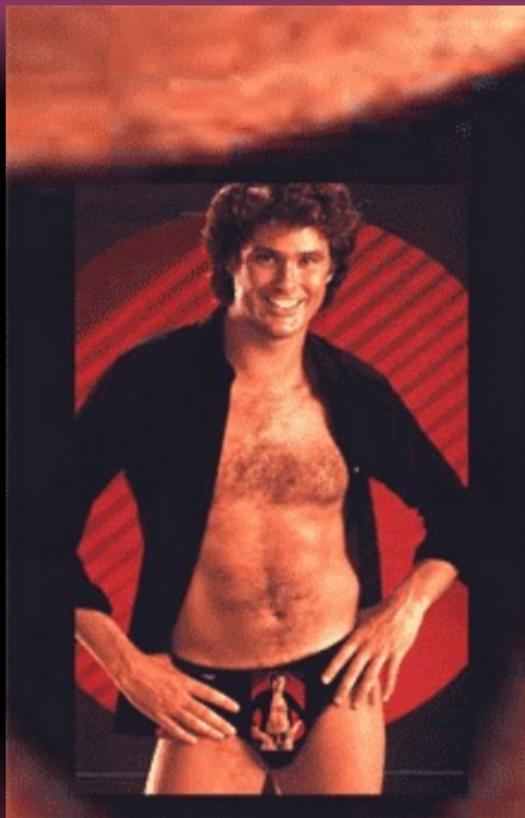


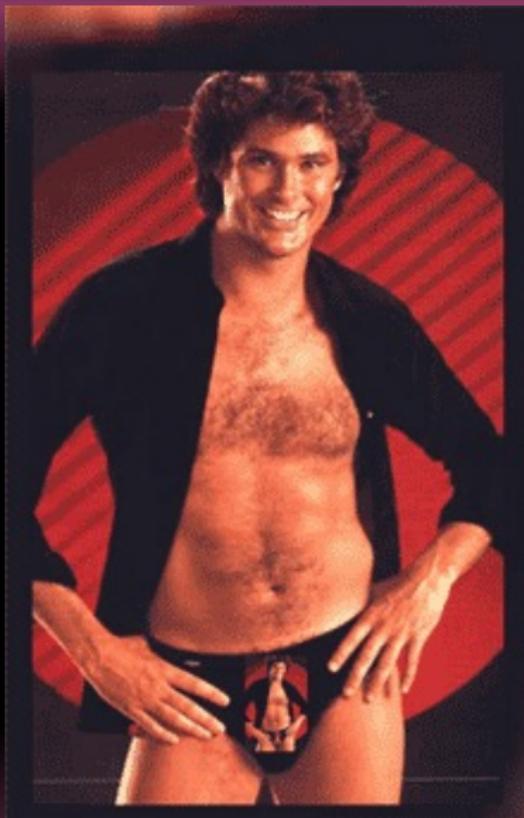


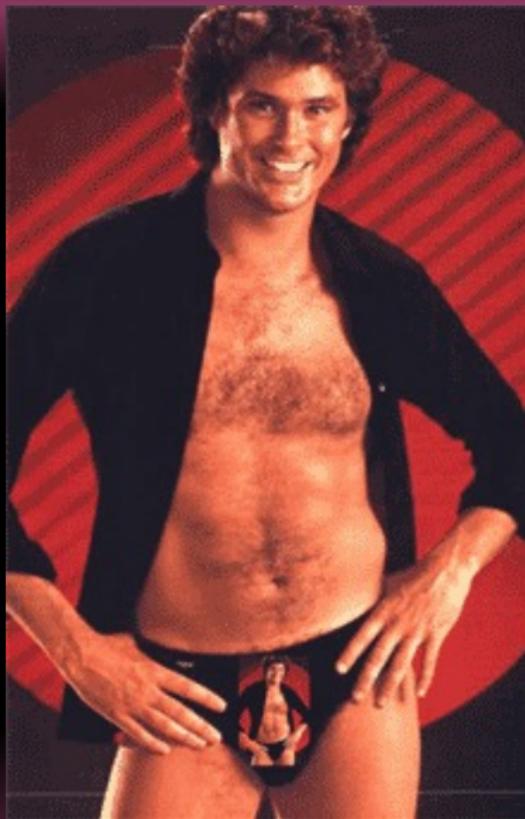




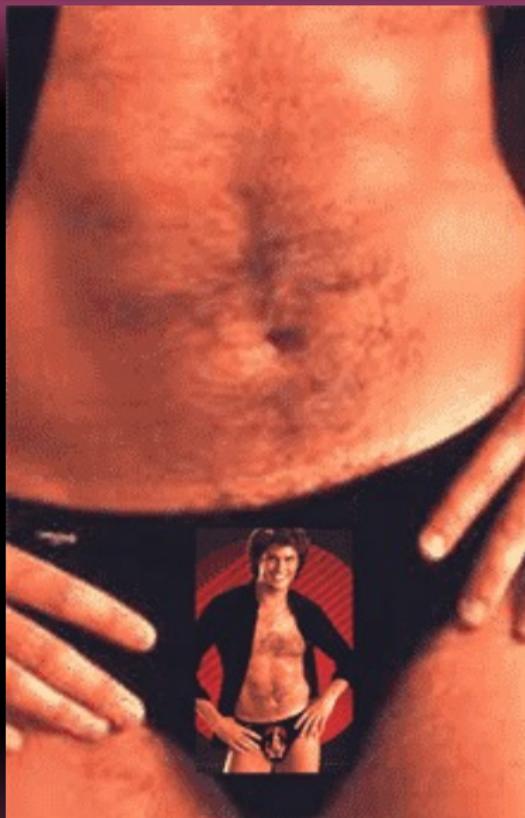






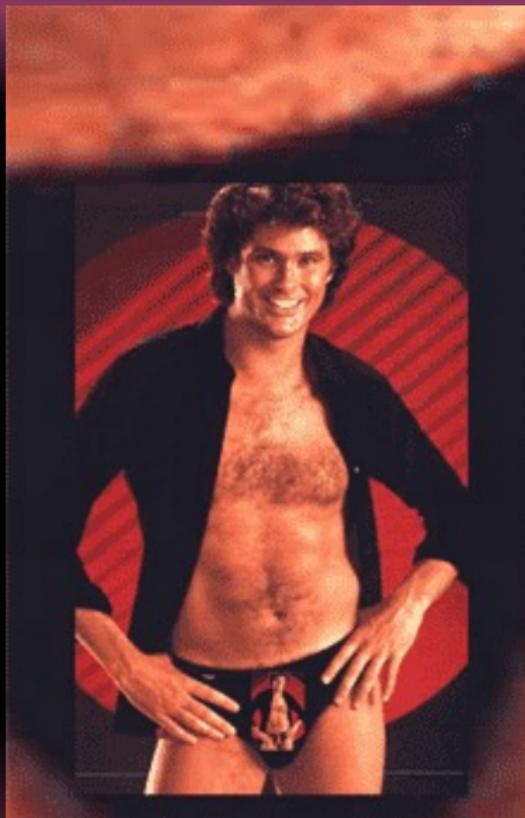


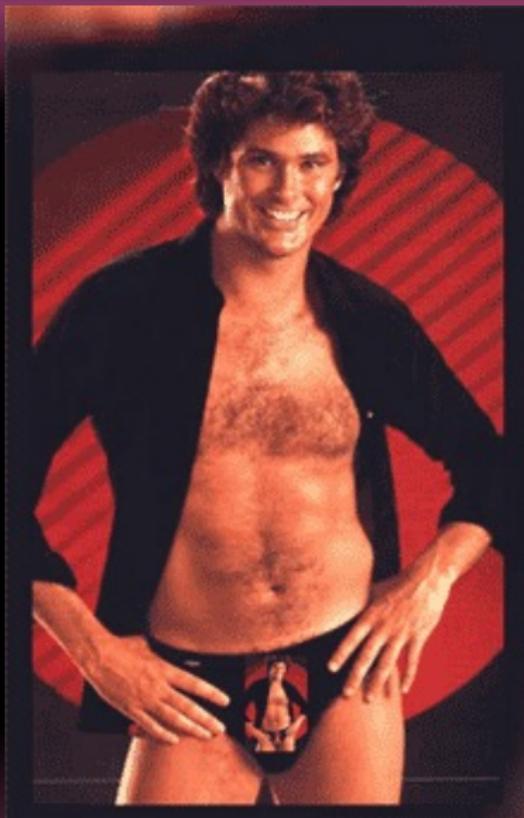












- induction
- induction mathématique (raisonnement par récurrence)
- fonction récursive et type récursif

- induction
- induction mathématique (raisonnement par récurrence)
- fonction récursive et type récursif

- induction
- induction mathématique (raisonnement par récurrence)
- fonction récursive et type récursif

Le retour de Max



```
def max2(a,b) :  
    return a if a >= b else b
```

```
def maxi(xs) :  
    assert xs != [], "Liste vide !"  
    if len(xs) == 1 :  
        return xs[0]  
    return max2( xs[0], maxi(xs[1:]) )
```

```
def max2(a,b) :  
    return a if a >= b else b
```

```
def maxi(xs) :  
    assert xs != [], "Liste vide !"  
    if len(xs) == 1 :  
        return xs[0]  
    return max2( xs[0], maxi(xs[1:]) )
```

```
In [54]: maxi("agfdjuol")
maxi <- ('agfdjuol',)
maxi <- ('gfdjuol',)
maxi <- ('fdjuol',)
maxi <- ('djuol',)
maxi <- ('juol',)
maxi <- ('uol',)
maxi <- ('ol',)
maxi <- ('l',)
maxi -> l
maxi -> o
maxi -> u
Out[54]: 'u'
```

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

... ???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

... ???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

... ???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ facteurs } a}$$

... ???

$$a^0 = 1, \quad a^n = a \times a^{n-1}$$

```
def pow1(a, n) :  
    assert n >= 0, "L'exposant doit être entier naturel"  
    if n == 0 :  
        return 1  
    else :  
        return a * pow1(a, n - 1)
```

```
pow1(2, 3)
```

```
2 * pow1(2, 2)
```

```
2 * (2 * pow1(2, 1))
```

```
2 * (2 * (2 * pow1(2, 0)))
```

```
2 * (2 * (2 * 1))
```

```
2 * (2 * 2)
```

```
2 * 4
```

```
8
```

Expansion Reduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2, 0)))
2 * (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Reduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2, 0)))
2 * (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Reduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion/Reduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion/Reduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Reduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
pow1(2, 3)
2 * pow1(2, 2)
2 * (2 * pow1(2, 1))
2 * (2 * (2 * pow1(2 , 0)))
2* (2 * (2 * 1))
2 * (2 * 2)
2 * 4
8
```

Expansion Réduction

```
def pow2(a, n) :  
    assert n >= 0, "L'exposant doit être entier naturel"  
    def accumule(k, acc) :  
        if k == 0 :  
            return acc  
        else :  
            return accumule(k - 1, a*acc)  
    return accumule(n, 1)
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1, 4)
accumule(0, 8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1, 4)
accumule(0, 8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1, 4)
accumule(0, 8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1,4)
accumule(0,8)
8
```

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1, 4)
accumule(0, 8)
```

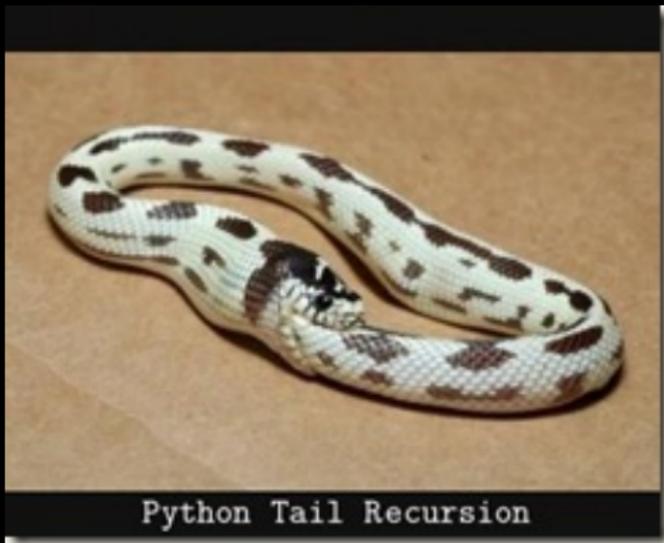
8

```
pow2(2, 3)
accumule(3, 1)
accumule(2, 2)
accumule(1, 4)
accumule(0, 8)
8
```



```
In [24]: pow1(2,3)
pow1 <- (2, 3)
  pow1 <- (2, 2)
    pow1 <- (2, 1)
      pow1 <- (2, 0)
        pow1 -> 1
          pow1 -> 2
            pow1 -> 4
              pow1 -> 8
Out[24]: 8
```

```
In [27]: pow2(2,3)
pow2 <- (2, 3)
accumule <- (3, 1)
  accumule <- (2, 2)
    accumule <- (1, 4)
      accumule <- (0, 8)
        accumule -> 8
          accumule -> 8
            accumule -> 8
              pow2 -> 8
                Out[27]: 8
```

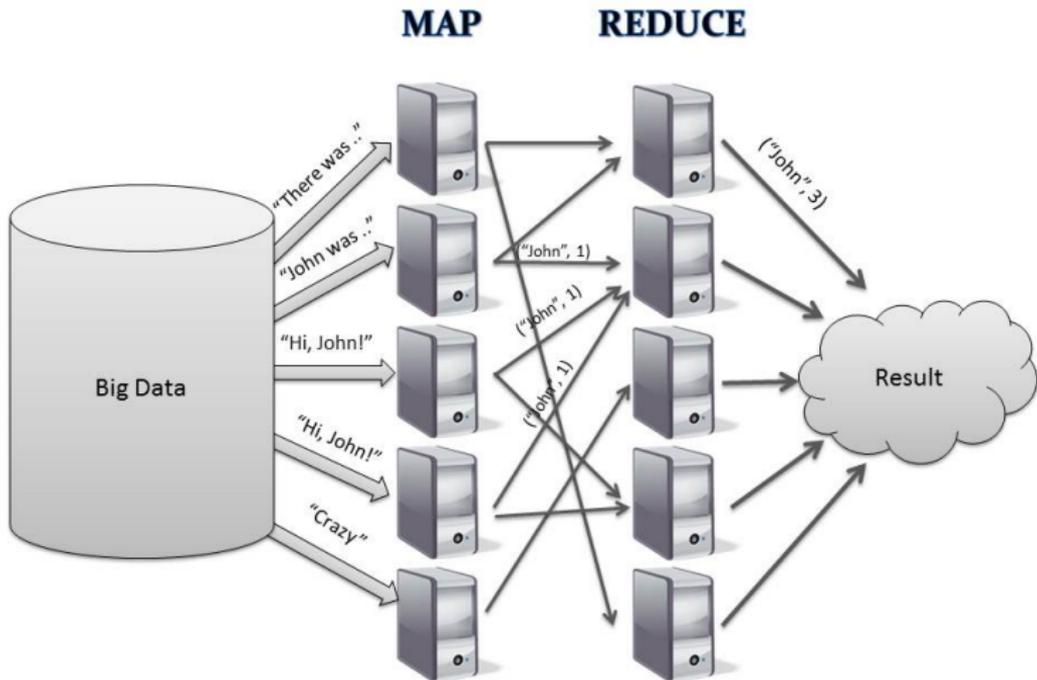


```
def pow3(a, n) :  
    assert n >= 0, "L'exposant doit être entier naturel"  
    accu = 1  
    k = 0  
    while k < n :  
        accu *= a  
        k += 1  
    return accu
```

```
def pow4(a, n) :  
    assert n >= 0, "L'exposant doit être un entier naturel"  
    accu = 1  
    for k in range(n) :  
        accu = a*accu  
    return accu
```

```
from functools import reduce

def pow5(a, n):
    return reduce(lambda accu, k : a*accu, range(n), 1)
```



Le secret de la réduction



Le secret de la réduction

$$f(f(f(\dots f(f(\text{accu0}, x_0), x_1)\dots), x_{n-1}), x_n)$$

```
def réduit(f, xs, accu0) :  
    accu = accu0  
    for x in xs :  
        accu = f(accu, x)  
    return accu
```

```
reduce(f, xs, accu0)
```

Le secret de la réduction

$$f(f(f(\dots f(f(\text{accu0}, x_0), x_1)\dots), x_{n-1}), x_n)$$

```
def réduit(f, xs, accu0) :  
    accu = accu0  
    for x in xs :  
        accu = f(accu, x)  
    return accu
```

```
reduce(f, xs, accu0)
```

Le secret de la réduction

$$f(f(f(\dots f(f(\text{accu0}, x_0), x_1)\dots), x_{n-1}), x_n)$$

```
def reduit(f, xs, accu0) :  
    accu = accu0  
    for x in xs :  
        accu = f(accu, x)  
    return accu
```

```
reduce(f, xs, accu0)
```

Le retour de Max



```
maxi = lambda xs: reduce(lambda max_tmp, x: x if x > max_tmp else max_tmp, xs)
```

```
max2 = lambda a, b: a if a >= b else b
```

```
maxi = lambda xs: reduce(max2, xs)
```

```
def max2(a,b) :  
    return a if a >= b else b
```

```
def maxi(xs) :  
    return reduce(max2, xs)
```

```
maxi = lambda xs: reduce(lambda max_tmp, x: x if x > max_tmp else max_tmp, xs)
```

```
max2 = lambda a, b: a if a >= b else b
```

```
maxi = lambda xs: reduce(max2, xs)
```

```
def max2(a,b) :  
    return a if a >= b else b
```

```
def maxi(xs) :  
    return reduce(max2, xs)
```

```
maxi = lambda xs: reduce(lambda max_tmp, x: x if x > max_tmp else max_tmp, xs)
```

```
max2 = lambda a, b: a if a >= b else b
```

```
maxi = lambda xs: reduce(max2, xs)
```

```
def max2(a,b) :  
    return a if a >= b else b
```

```
def maxi(xs) :  
    return reduce(max2, xs)
```

```
In [23]: maxi([1,12,5,47,86,9,4])
```

```
Out[23]: 86
```

```
In [24]: maxi([])
```

```
TypeError: reduce() of empty sequence with no initial value
```

Défi

Construisez une fonction prend une liste d'entiers et renvoie le produit des carrés des nombres pairs de cette liste. Une réponse en une petite ligne serait appréciée...

Défi

Construisez une fonction compose qui renvoie la composée d'une suite de fonctions :

```
In [35]: compose(maximum,longueurs,eclate)("Trois deux un")  
Out[35]: 5
```

Défi

Construisez une fonction prend une liste d'entiers et renvoie le produit des carrés des nombres pairs de cette liste. Une réponse en une petite ligne serait appréciée...

Défi

Construisez une fonction compose qui renvoie la composée d'une suite de fonctions :

```
In [35]: compose(maximum,longueurs,eclate)("Trois deux un")  
Out[35]: 5
```

```
In [29]: %timeit pow5(17,100)
100000 loops, best of 3: 12.3  $\mu$ s per loop
```

```
In [30]: %timeit pow4(17,100)
100000 loops, best of 3: 6.67  $\mu$ s per loop
```

```
In [31]: %timeit pow3(17,100)
100000 loops, best of 3: 9.24  $\mu$ s per loop
```

```
In [32]: %timeit pow2(17,100)
10000 loops, best of 3: 18.6  $\mu$ s per loop
```

```
In [33]: %timeit pow1(17,100)
10000 loops, best of 3: 17.8  $\mu$ s per loop
```

```
def pow6(a, n):  
    if n == 0 :  
        return 1  
    if n == 1 :  
        return a  
    if n % 2 == 0 :  
        return pow6(a*a, n//2)  
    return a * pow6(a, n-1)
```

```
In [42]: %timeit pow6(17,100)  
100000 loops, best of 3: 2.35  $\mu$ s per loop
```

```
def pow6(a, n):  
    if n == 0 :  
        return 1  
    if n == 1 :  
        return a  
    if n % 2 == 0 :  
        return pow6(a*a, n//2)  
    return a * pow6(a, n-1)
```

```
In [42]: %timeit pow6(17,100)  
100000 loops, best of 3: 2.35  $\mu$ s per loop
```

```
In [45]: pow6(2,10)
pow6 <- (2, 10)
pow6 <- (4, 5)
pow6 <- (4, 4)
  pow6 <- (16, 2)
    pow6 <- (256, 1)
      pow6 -> 256
        pow6 -> 256
          pow6 -> 256
            pow6 -> 1024
              pow6 -> 1024
                Out[45]: 1024
```

Défi

Dérécursifiez la fonction `pow6`



OCaml

```
let rec pow a = function
  | 1 -> 1
  | n -> a * (pow a (n - 1)) ;;
```

```
let powt a n =
  let rec pow_iter acc k =
    if k == 0 then acc
    else pow_iter (a*acc) (k-1)
  in pow_iter 1 n ;;
```

```
# pow 17 100_000;;  
Stack overflow during evaluation (looping recursion?).  
  
# powt 17 100_000;;  
- : int = -573761023
```

Sommaire

- 1 Fonctions pures
- 2 Fonctions impures
- 3 lambda expressions

- 4 Composition
- 5 Récursion
- 6 Raisonement par récurrence**
- 7 Suites de Fibonacci

Pour prouver qu'une propriété P_n dépendant uniquement d'un paramètre n est vraie pour tout $n \geq n_0$, il faut vérifier que :

- P_{n_0} est vraie (on parle parfois d'initialisation) ;
- pour tout $n \geq n_0$, $P_n \rightarrow P_{n+1}$ (on parle parfois d'hérédité).

Pour prouver qu'une propriété P_n dépendant uniquement d'un paramètre n est vraie pour tout $n \geq n_0$, il faut vérifier que :

- P_{n_0} est vraie (on parle parfois d'initialisation) ;
- pour tout $n \geq n_0$, $P_n \rightarrow P_{n+1}$ (on parle parfois d'hérédité).

Tours de Hanoi – 1 Disque

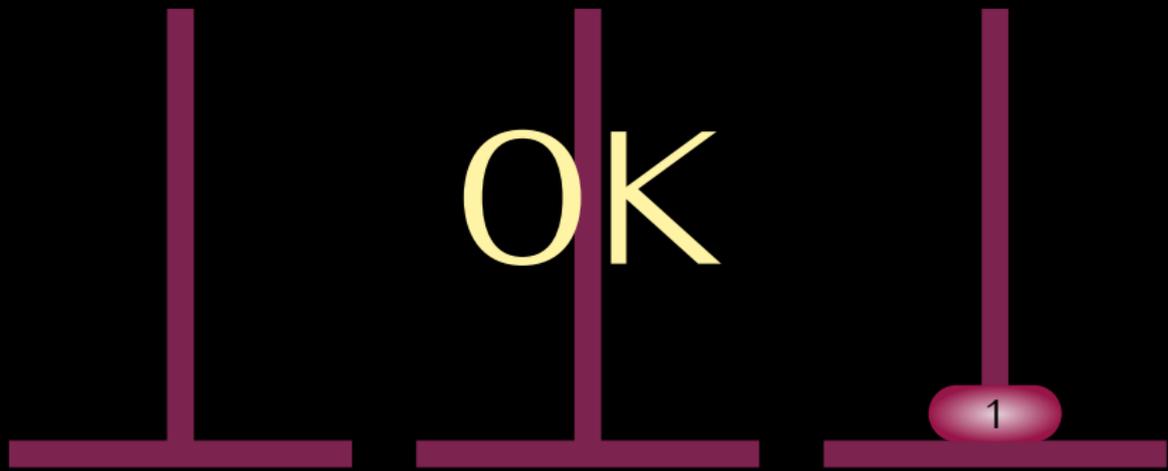


Tours de Hanoi – 1 Disque



On déplace le disque de la tige 1 vers la tige 3.

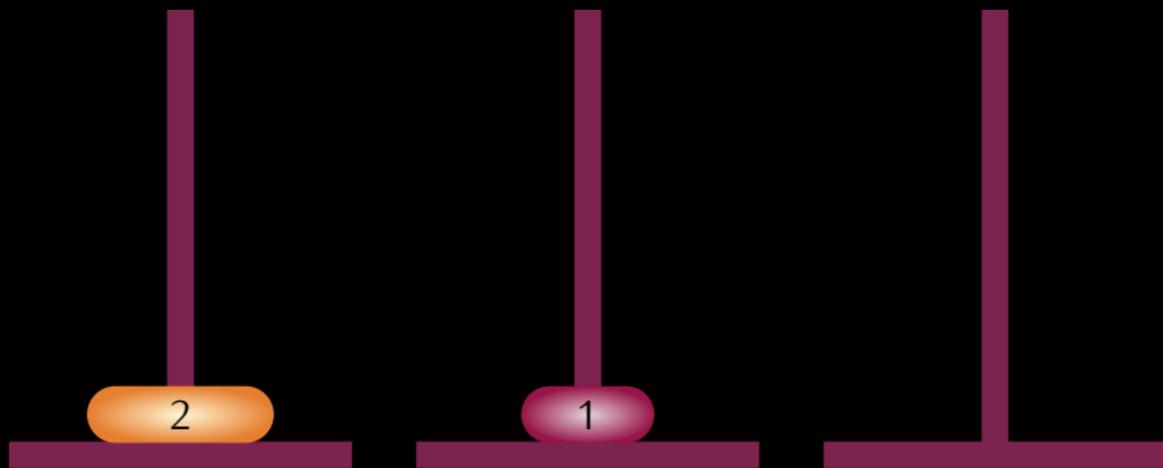
Tours de Hanoi – 1 Disque



Tours de Hanoi – 2 Disques

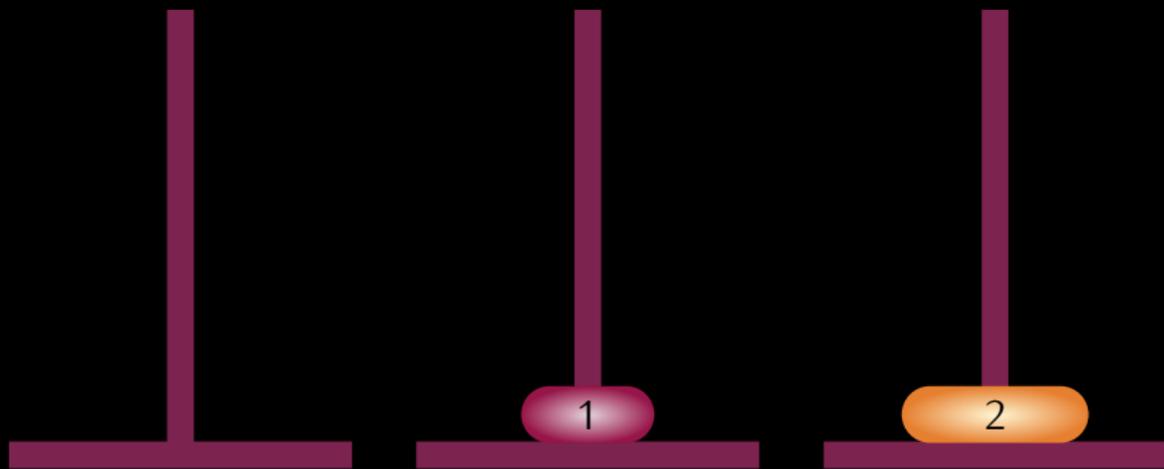


Tours de Hanoi – 2 Disques



On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 2 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 2 Disques



On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 2 Disques



Tours de Hanoi – 3 Disques

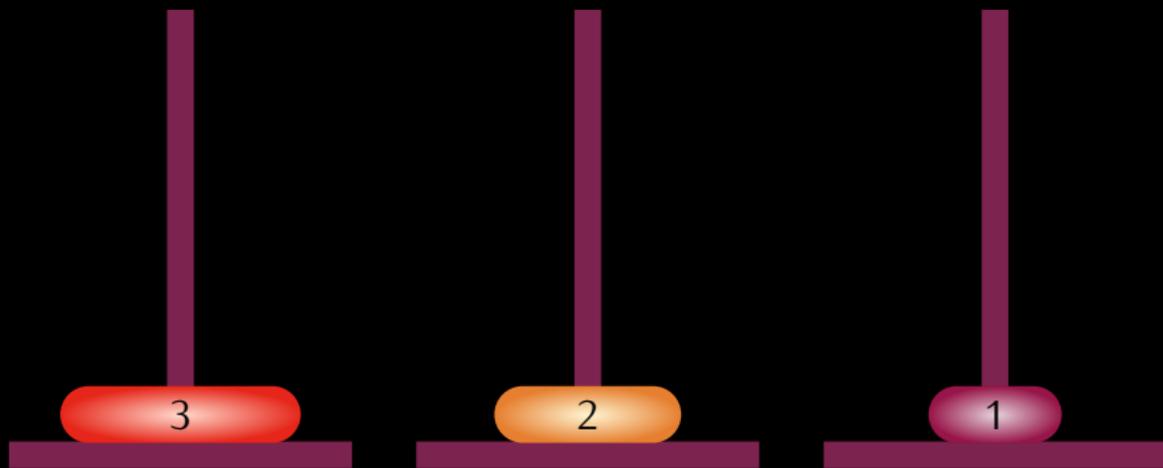


Tours de Hanoi – 3 Disques



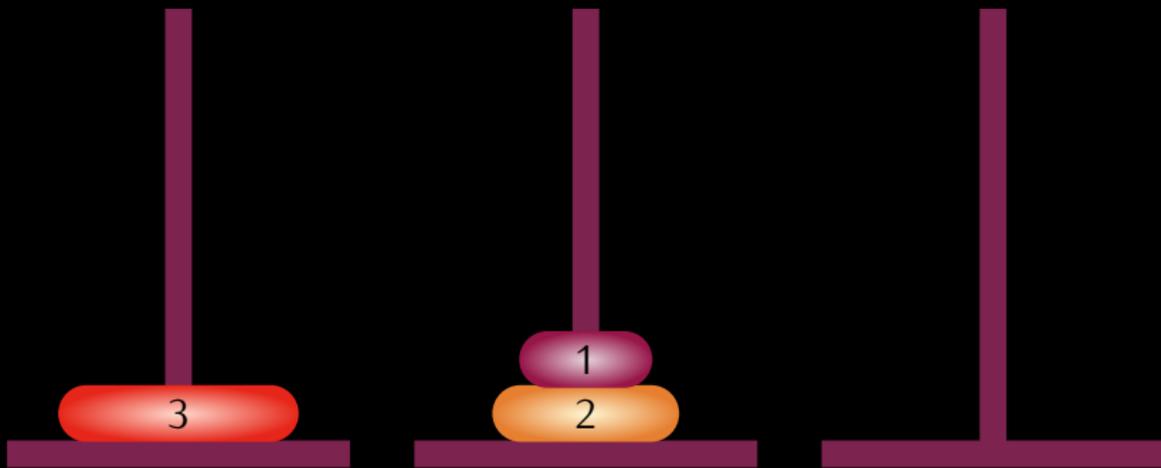
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 3 Disques



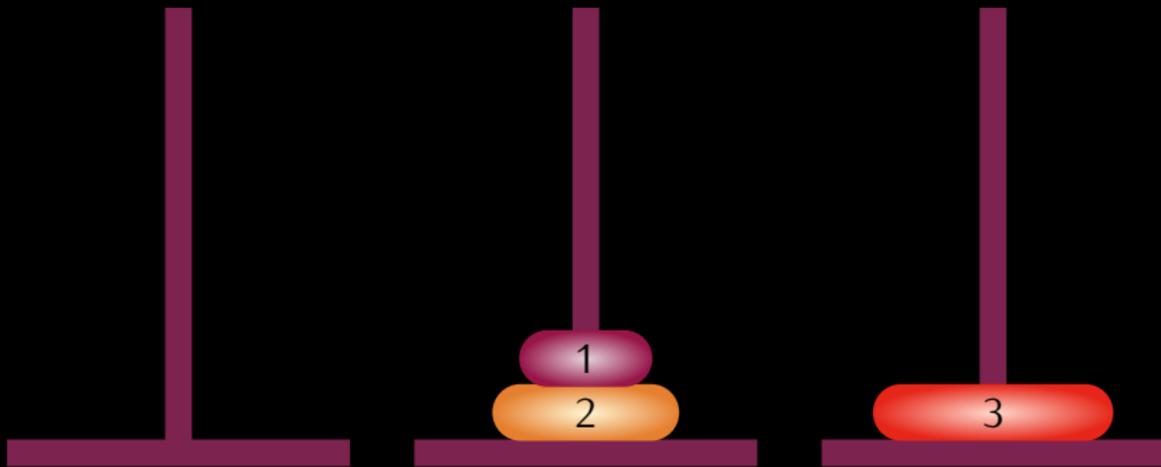
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 3 Disques



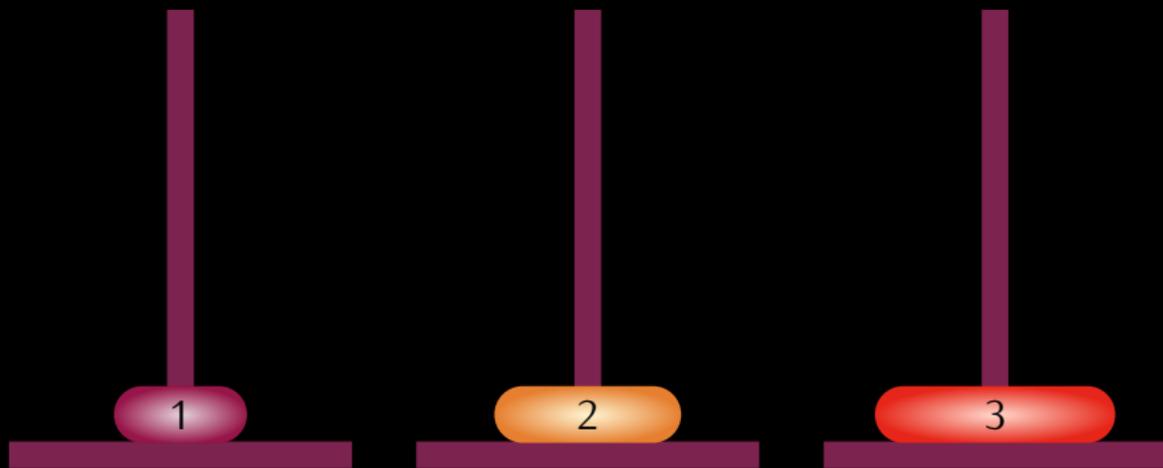
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 3 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 3 Disques



On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 3 Disques



On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 3 Disques



On déplace le disque de la tige 1 vers la tige 3.

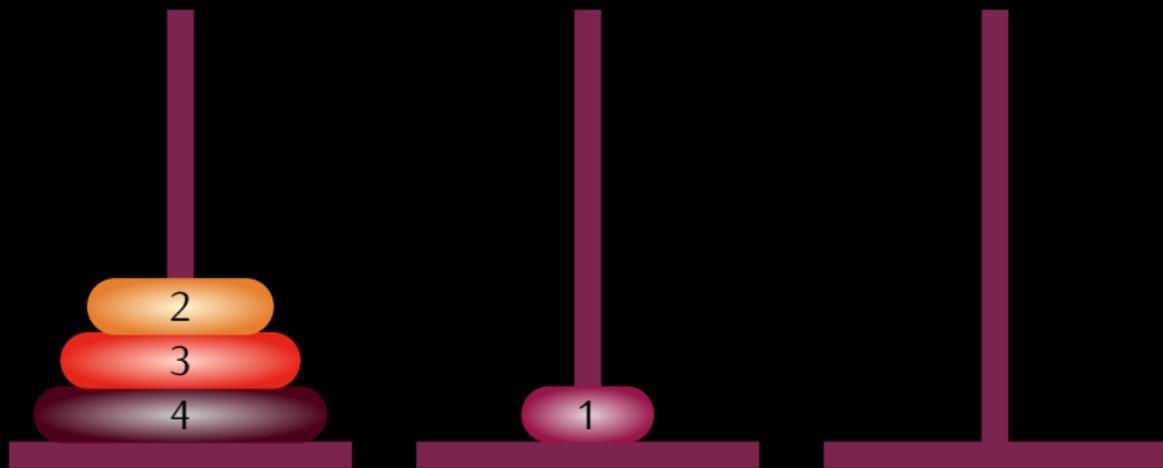
Tours de Hanoi – 3 Disques



Tours de Hanoi – 4 Disques

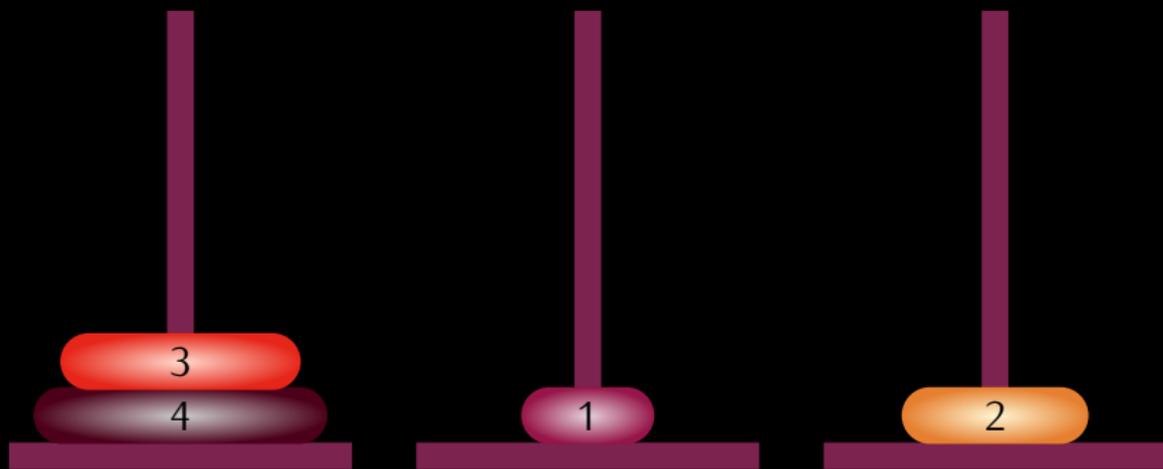


Tours de Hanoi – 4 Disques



On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



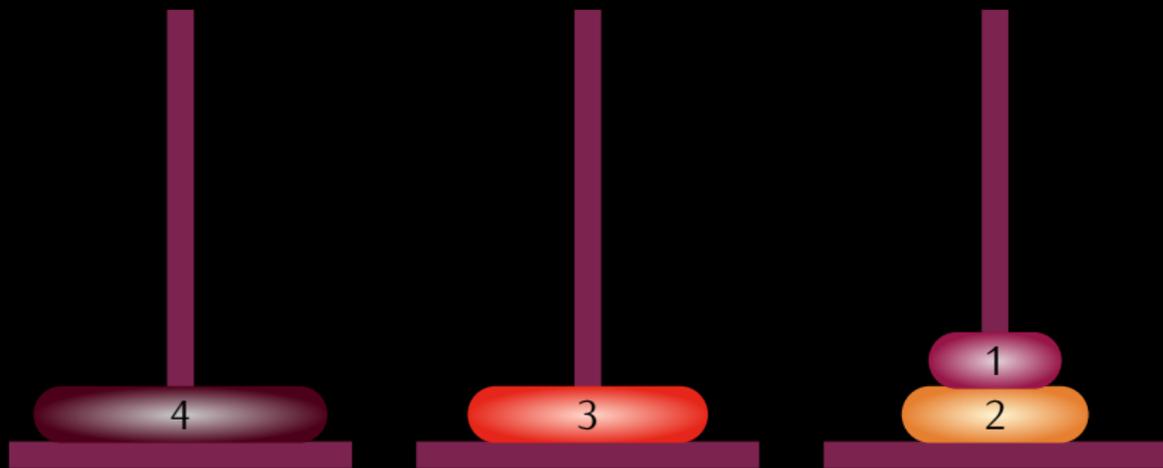
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



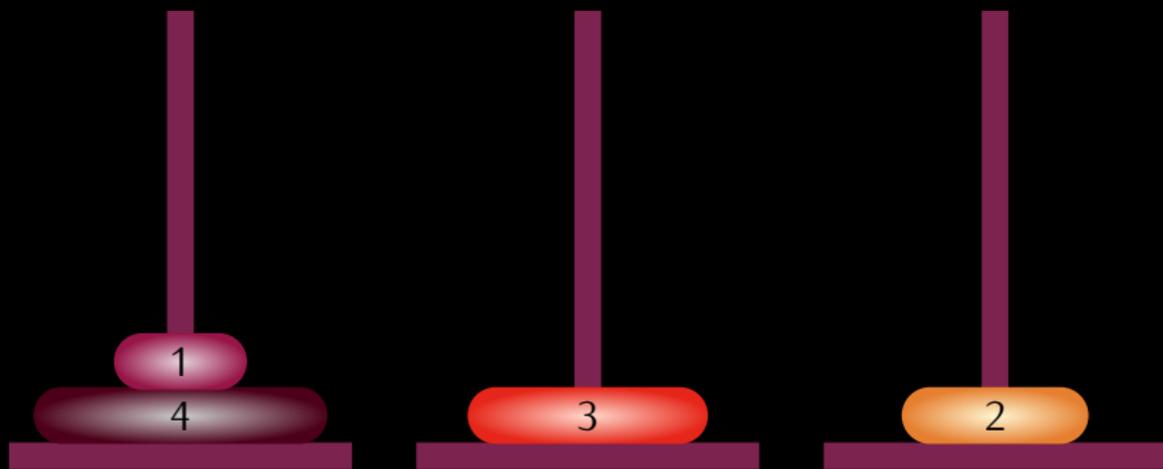
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



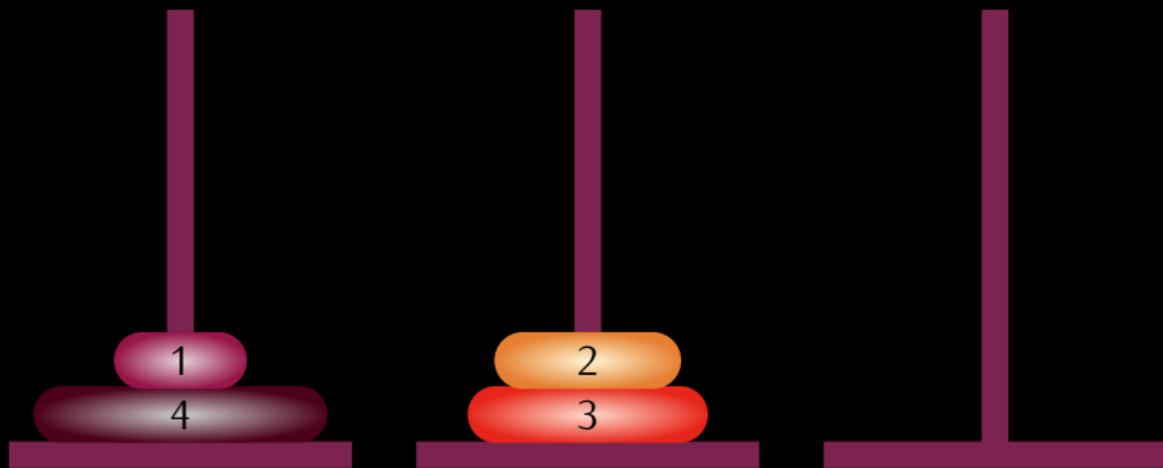
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



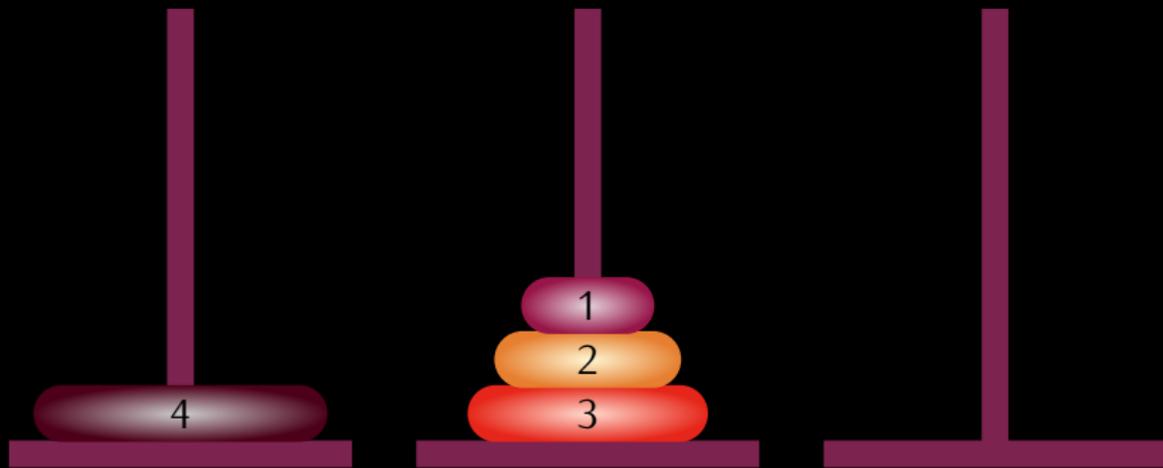
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 4 Disques



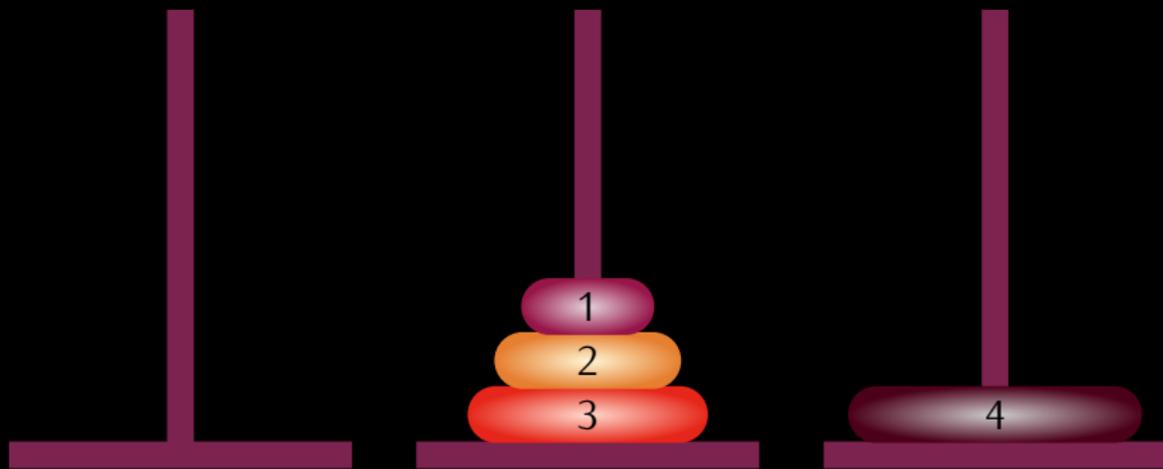
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 4 Disques



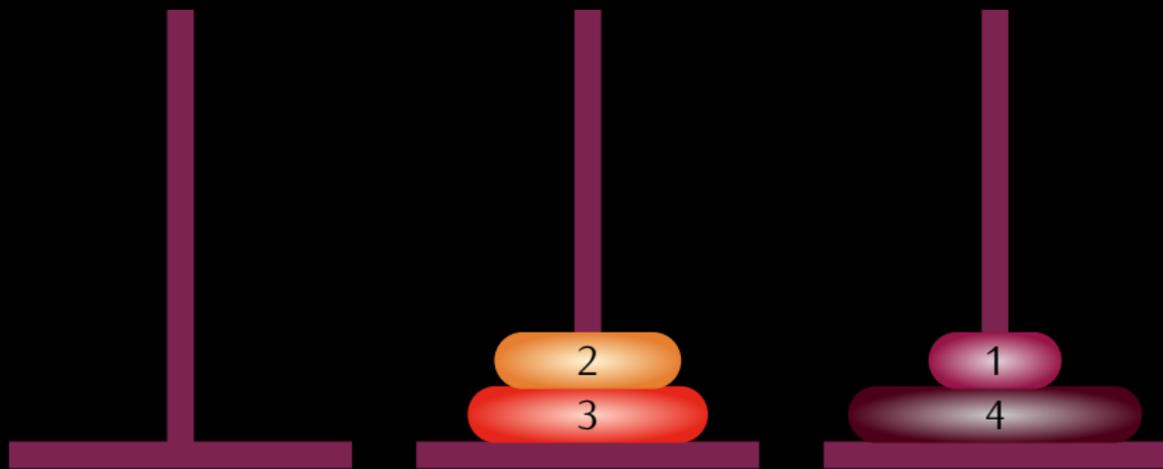
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



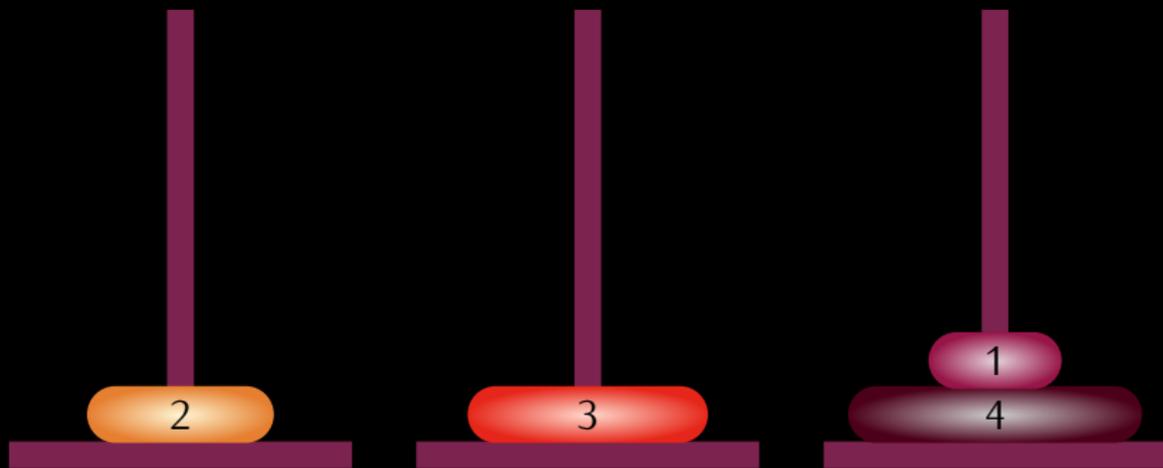
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



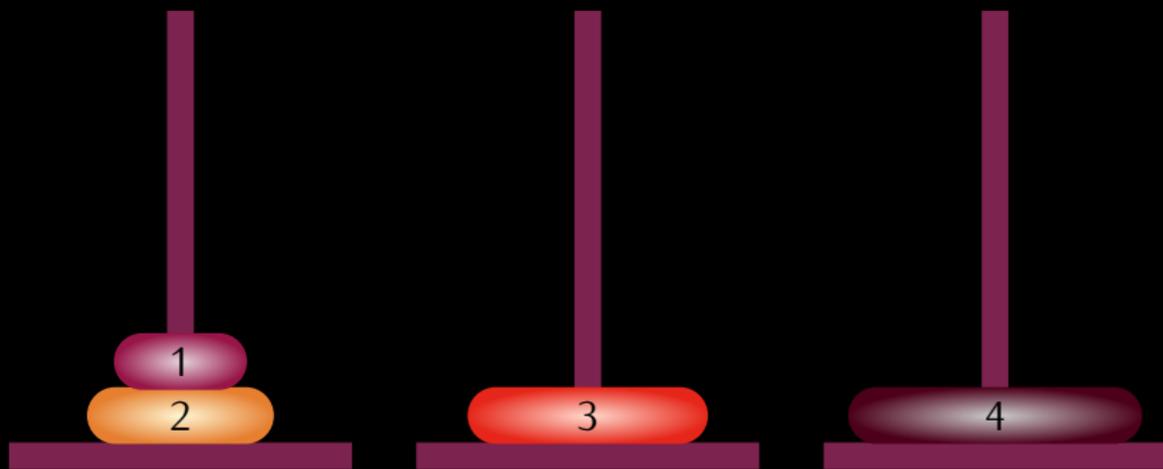
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 4 Disques



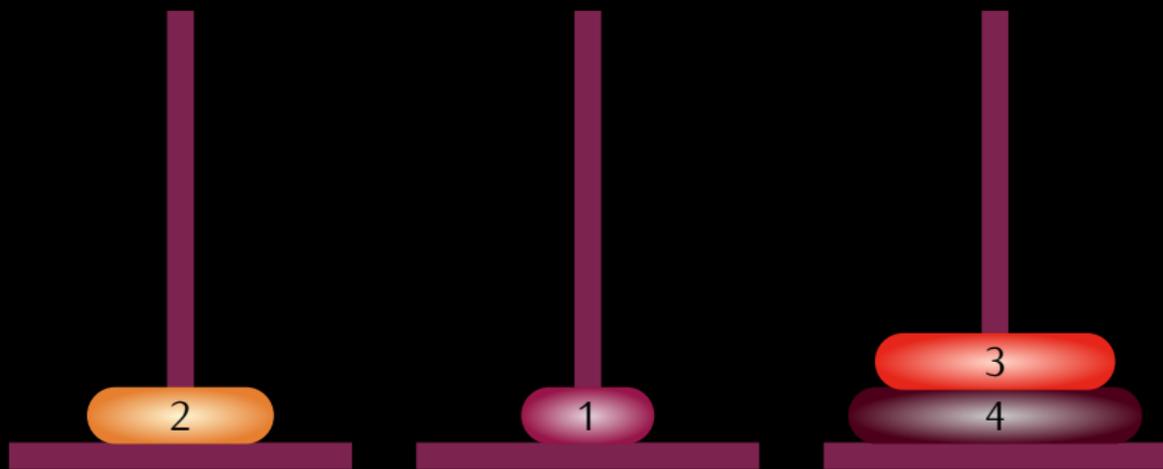
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 4 Disques



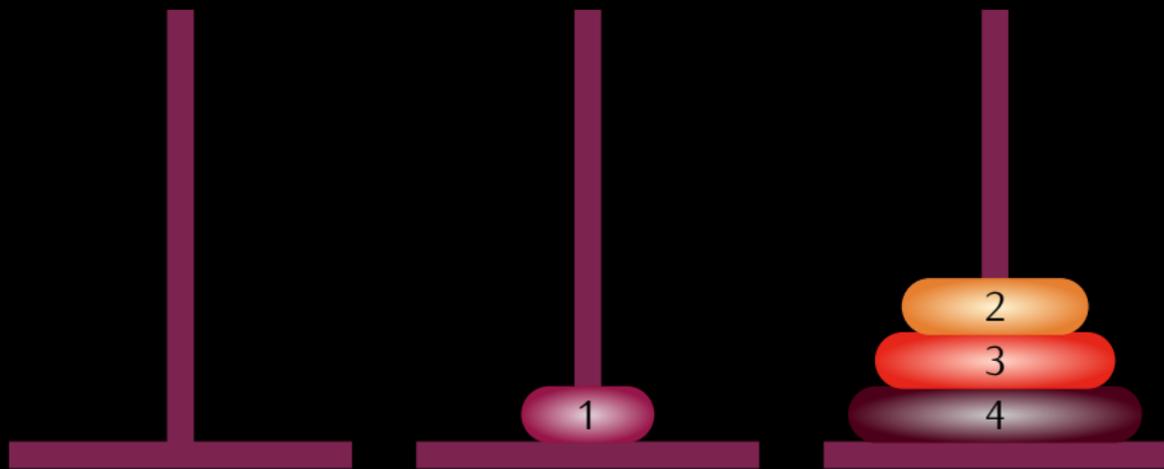
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 4 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 4 Disques



On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 4 Disques



Tours de Hanoi – 5 Disques

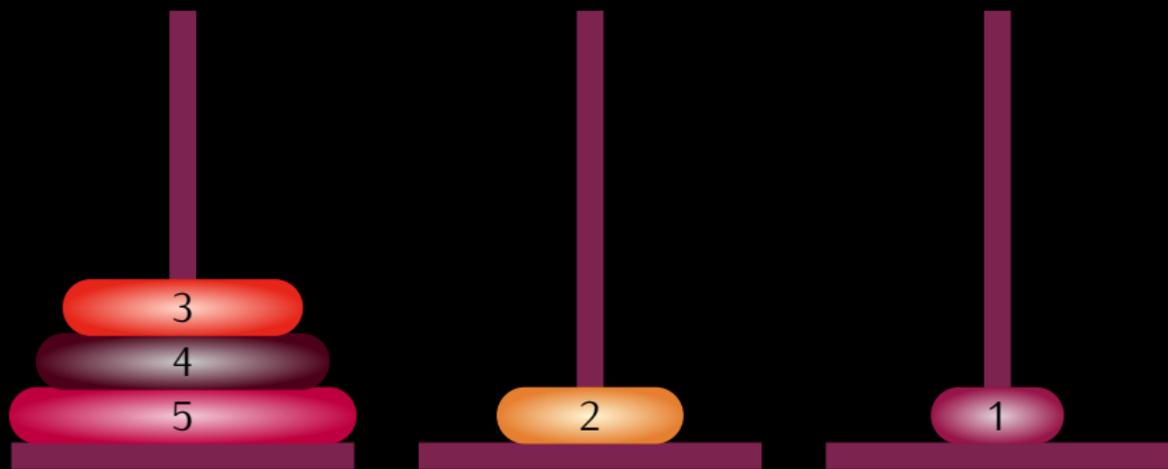


Tours de Hanoi – 5 Disques



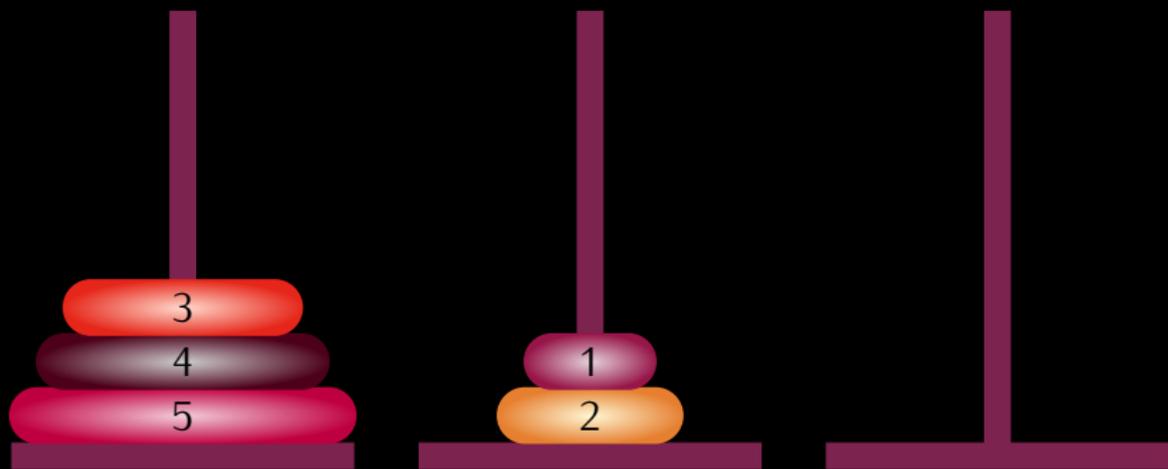
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



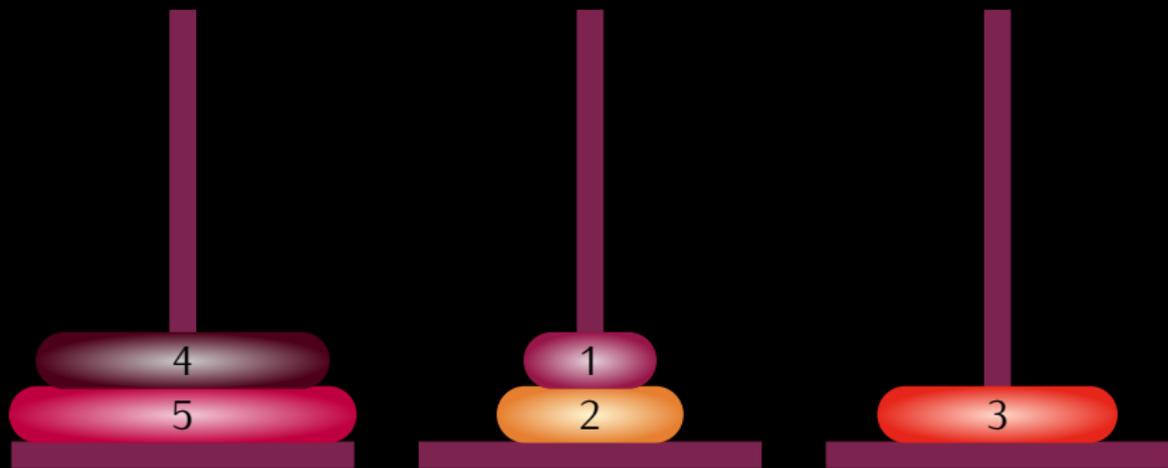
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



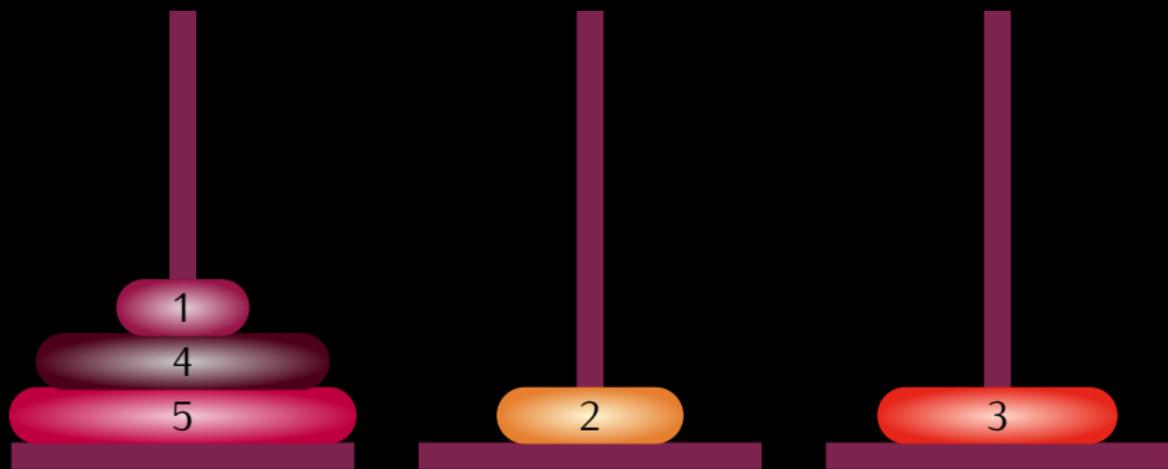
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



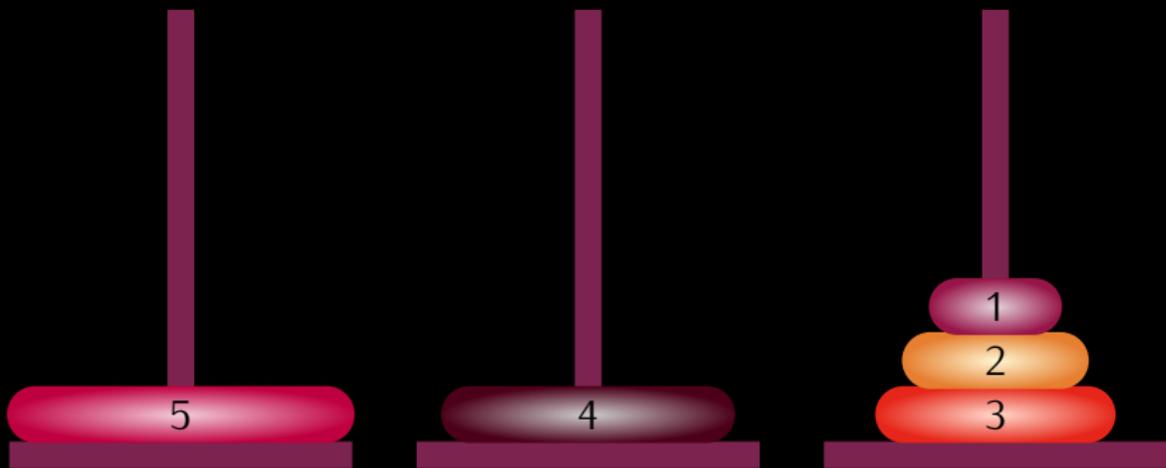
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



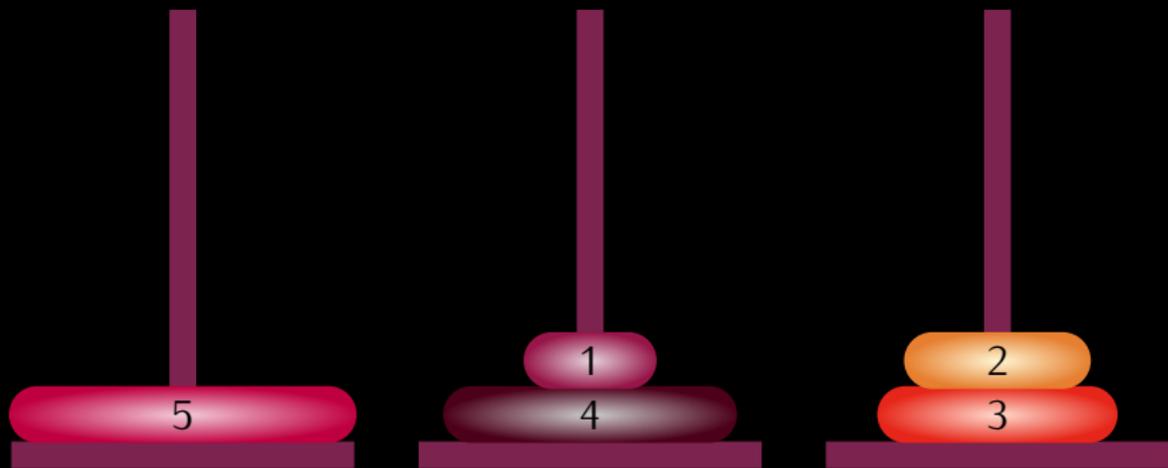
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



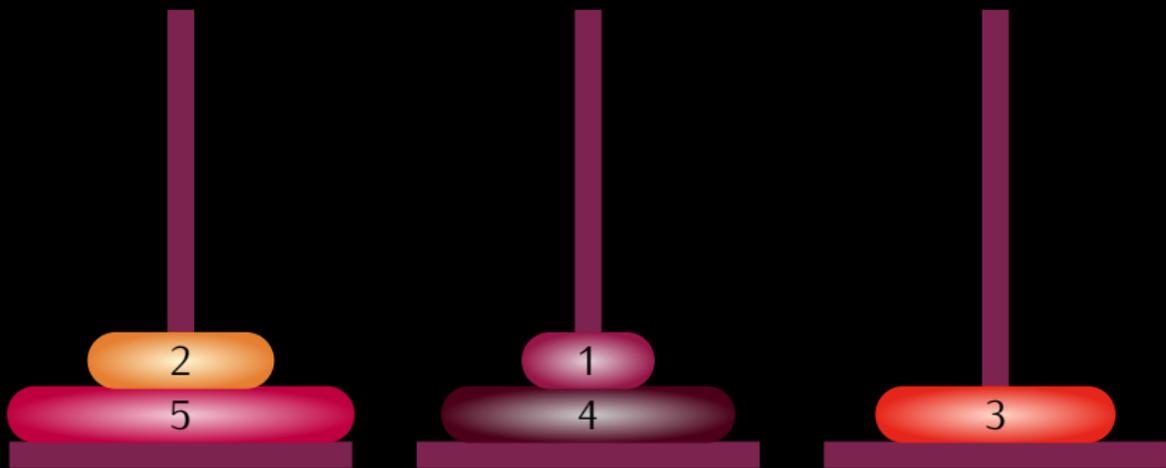
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



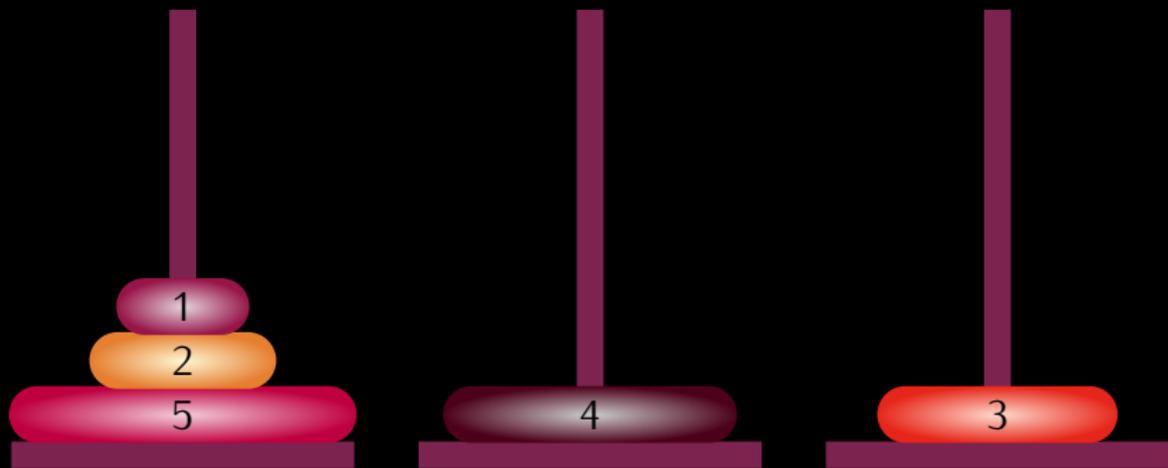
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



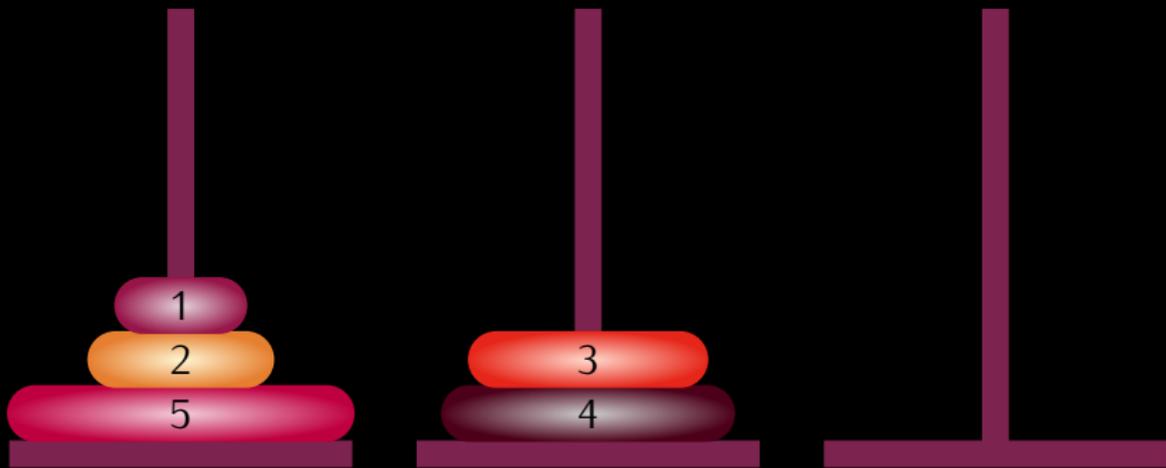
On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 5 Disques



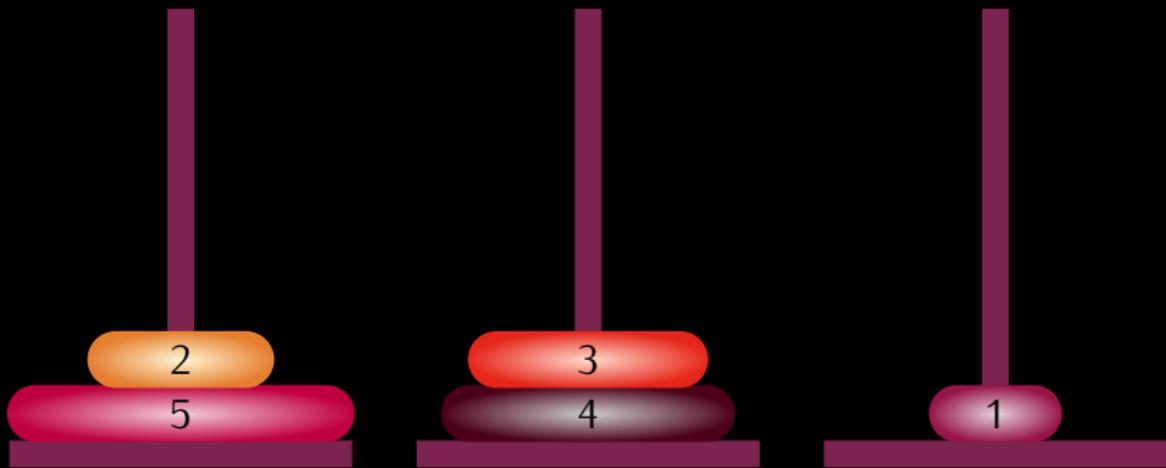
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



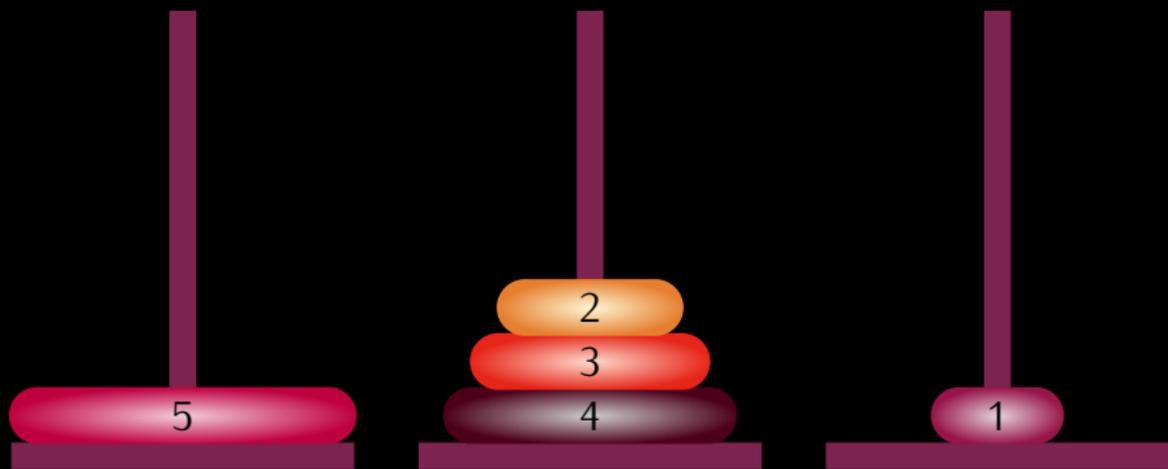
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



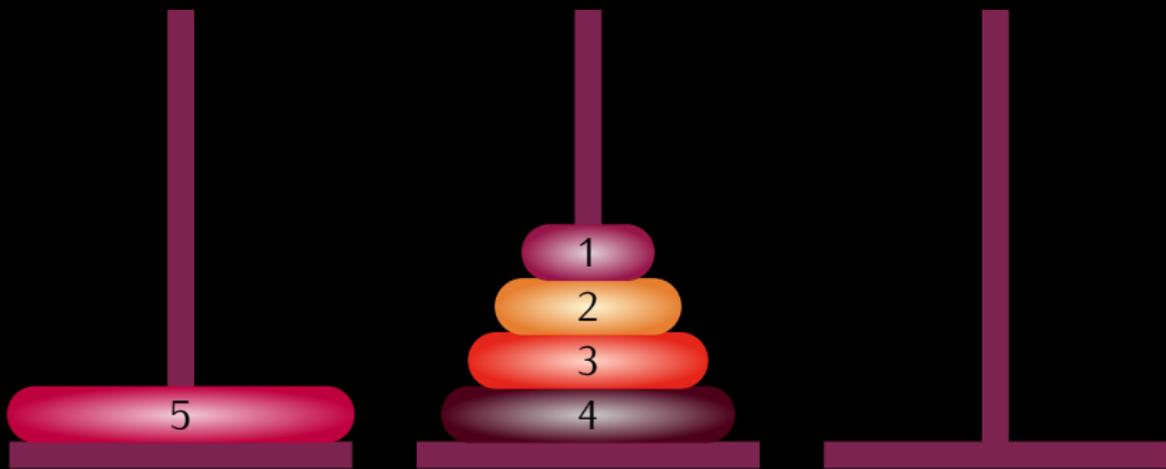
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



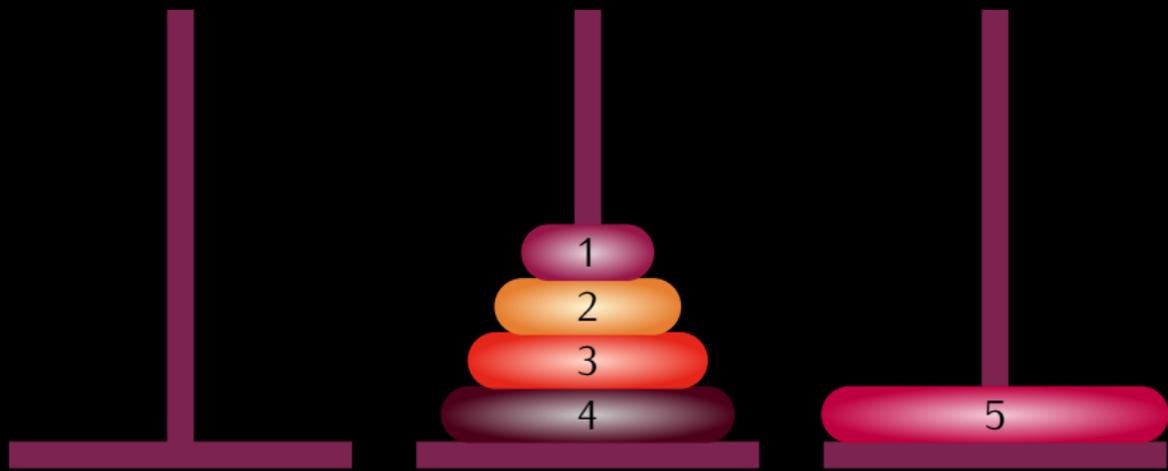
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



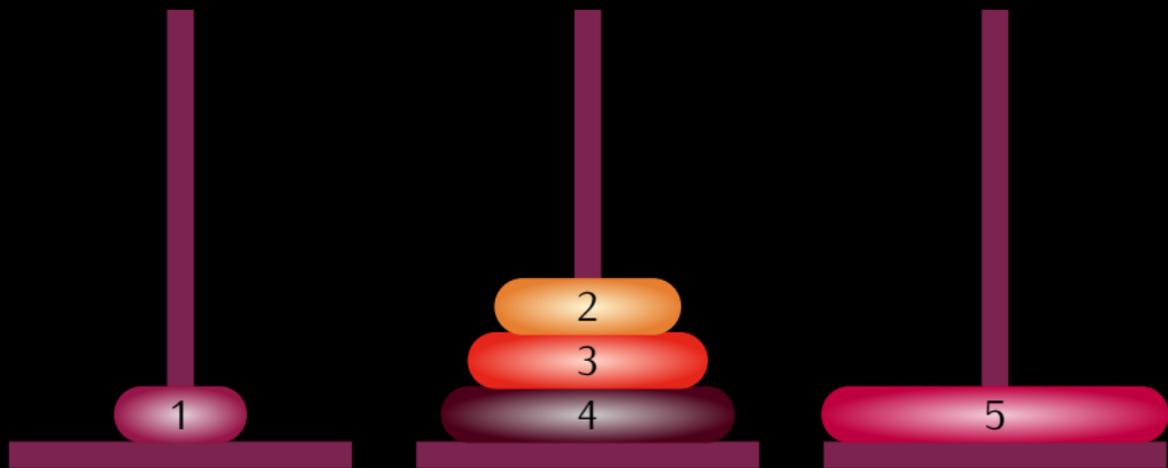
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



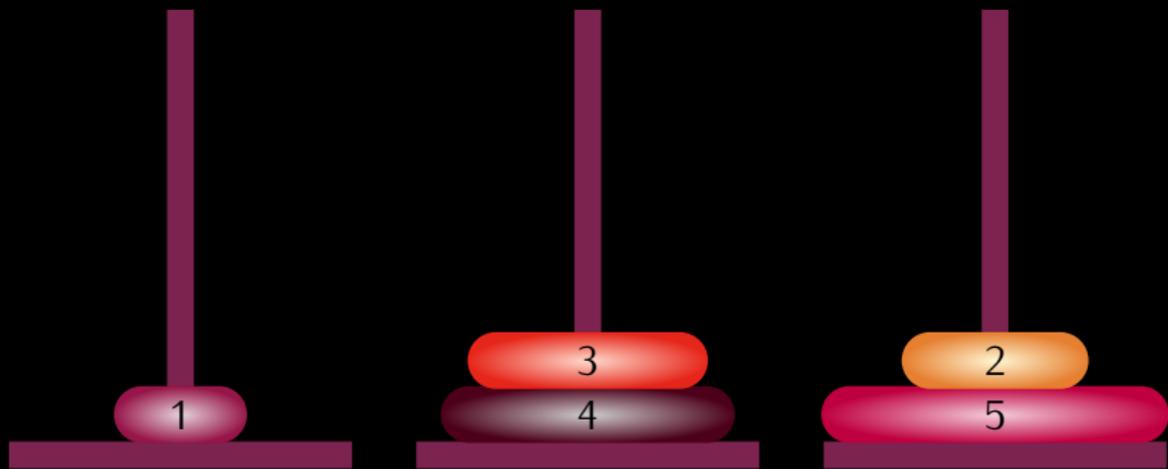
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



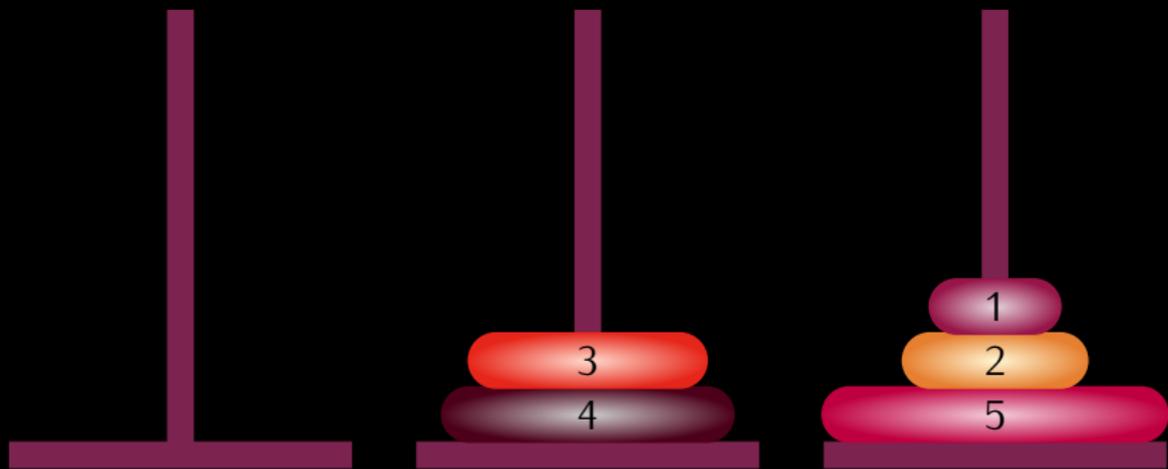
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



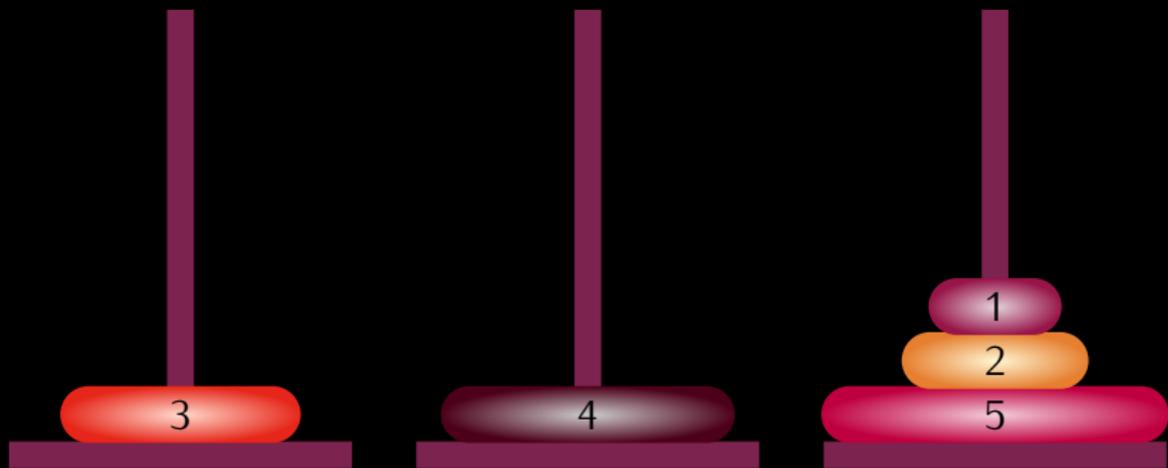
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



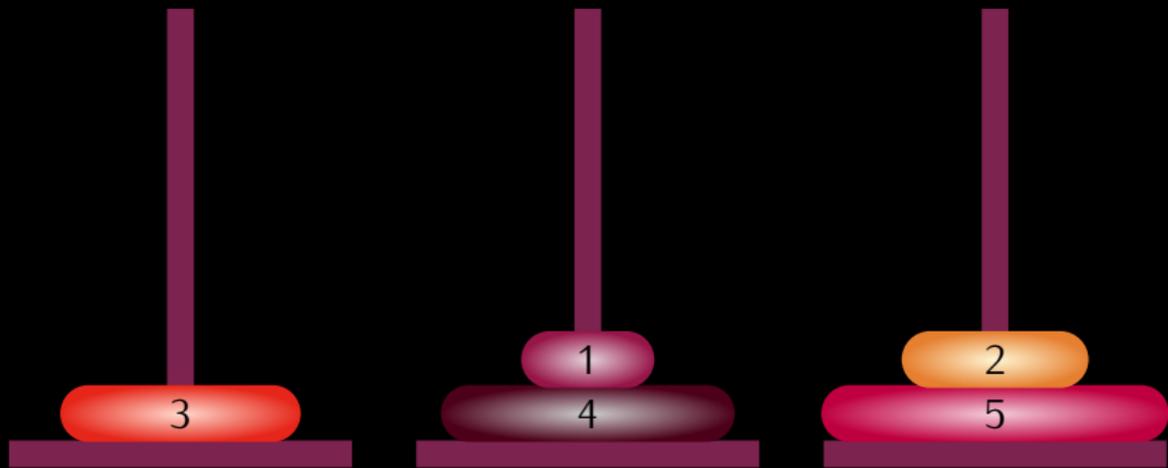
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



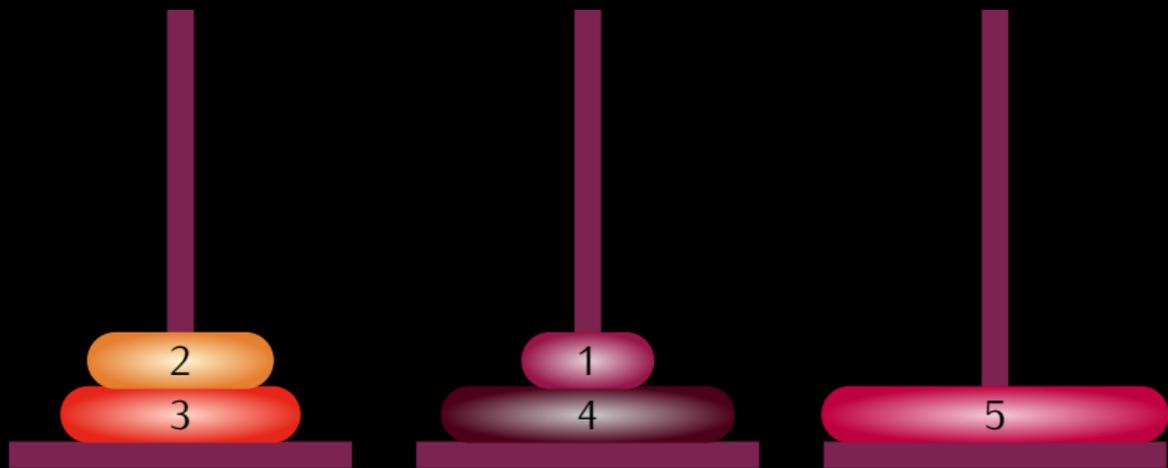
On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



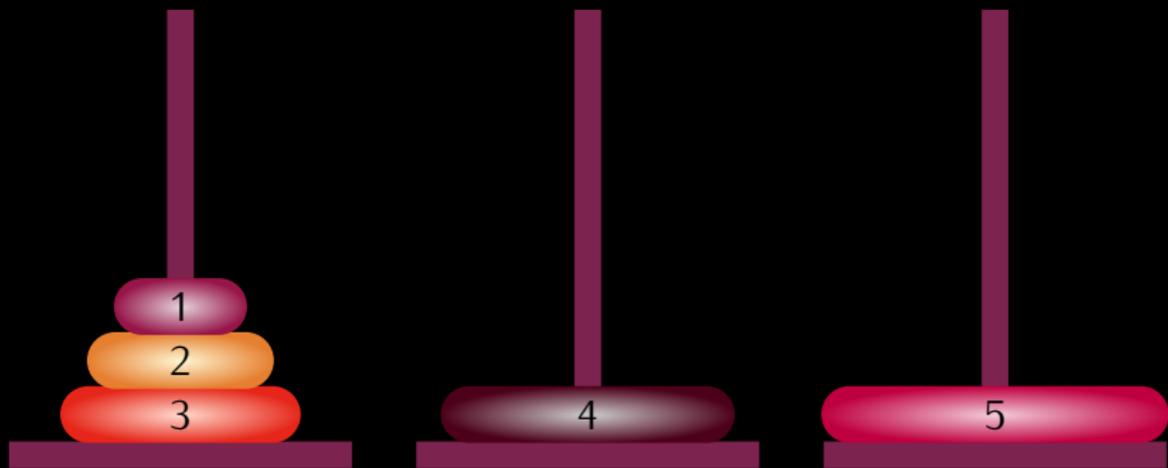
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 3 vers la tige 1.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



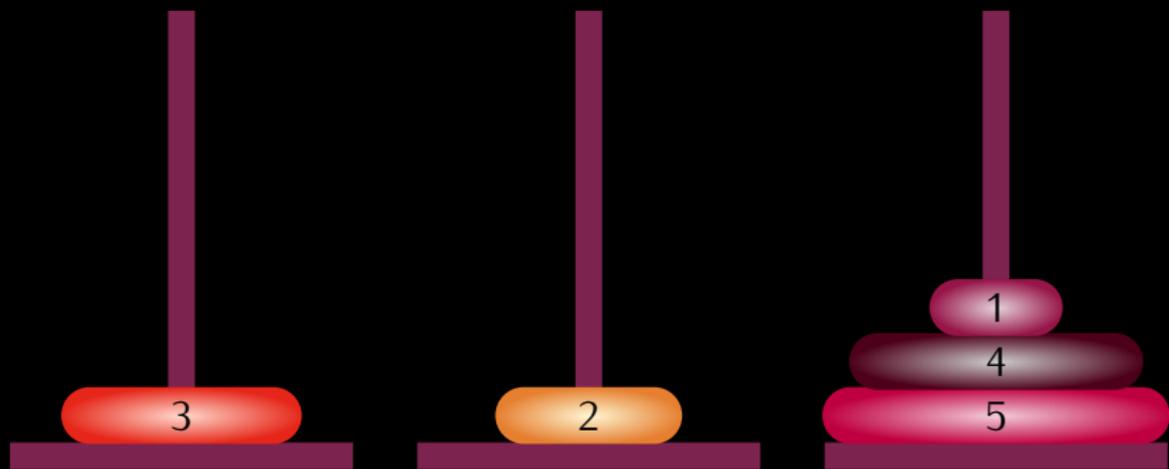
On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



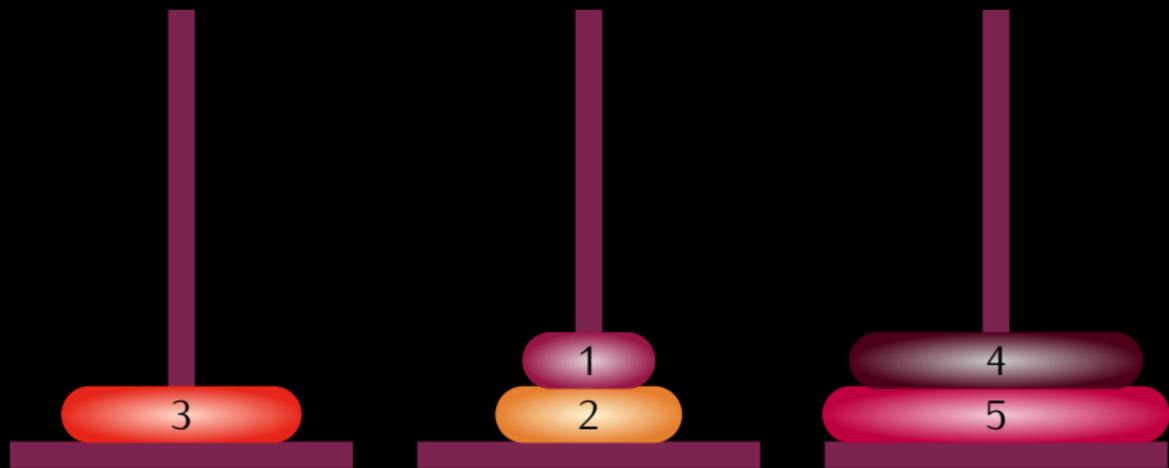
On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



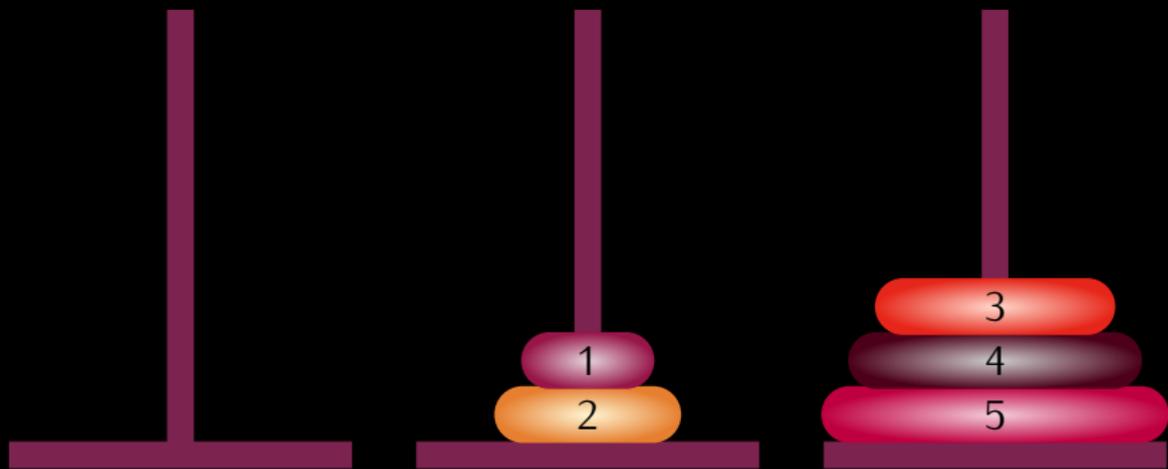
On déplace le disque de la tige 1 vers la tige 2.

Tours de Hanoi – 5 Disques



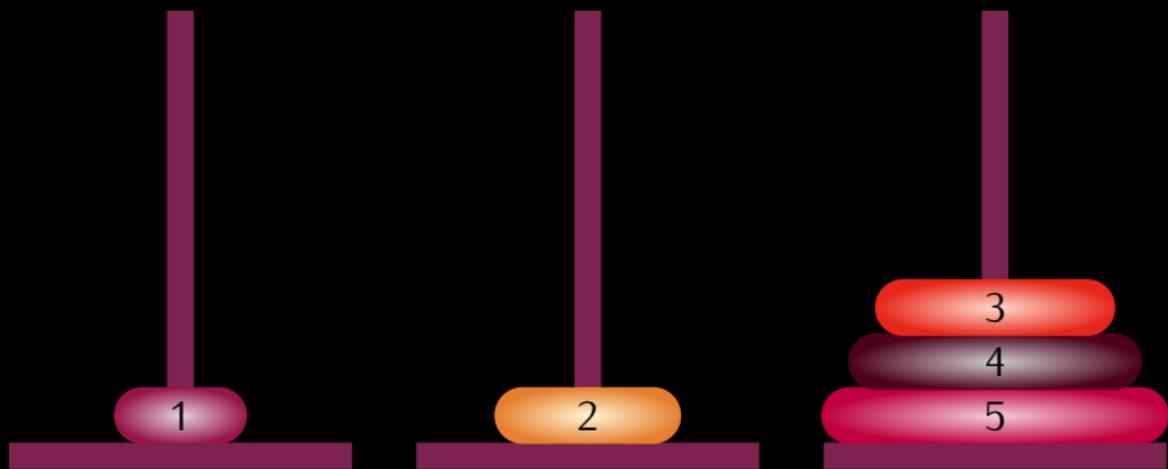
On déplace le disque de la tige 3 vers la tige 2.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 2 vers la tige 1.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 2 vers la tige 3.

Tours de Hanoi – 5 Disques



On déplace le disque de la tige 1 vers la tige 3.

Tours de Hanoi – 5 Disques



Tour de Hanoi – 6 Disques



Défi

Déterminez le nombre de mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Déterminez le nombre de mouvements effectués pour déplacer n disques des Tours de Hanoï.

Créez une fonction qui affiche les mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

Déterminez le nombre de mouvements effectués pour déplacer n disques des Tours de Hanoï.

Défi

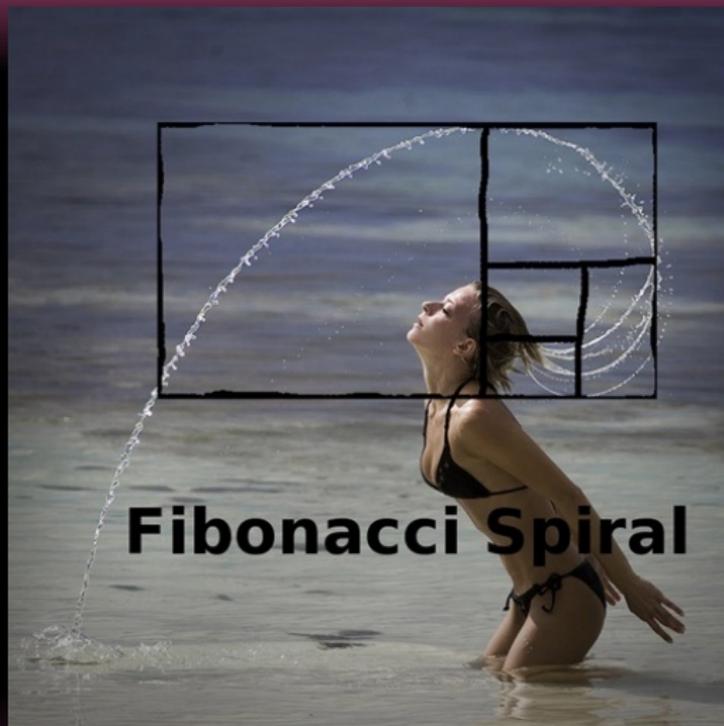
Créez une fonction qui affiche les mouvements effectués pour déplacer n disques des Tours de Hanoï.

```
In [52]: hanoi(3)
On déplace le disque de la tige 1 vers la tige 3
On déplace le disque de la tige 1 vers la tige 2
On déplace le disque de la tige 3 vers la tige 2
On déplace le disque de la tige 1 vers la tige 3
On déplace le disque de la tige 2 vers la tige 1
On déplace le disque de la tige 2 vers la tige 3
On déplace le disque de la tige 1 vers la tige 3
```

Sommaire

- 1 Fonctions pures
- 2 Fonctions impures
- 3 lambda expressions

- 4 Composition
- 5 Récursion
- 6 Raisonnement par récurrence
- 7 Suites de Fibonacci



```
def F(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return F(n - 1) + F(n - 2)
```

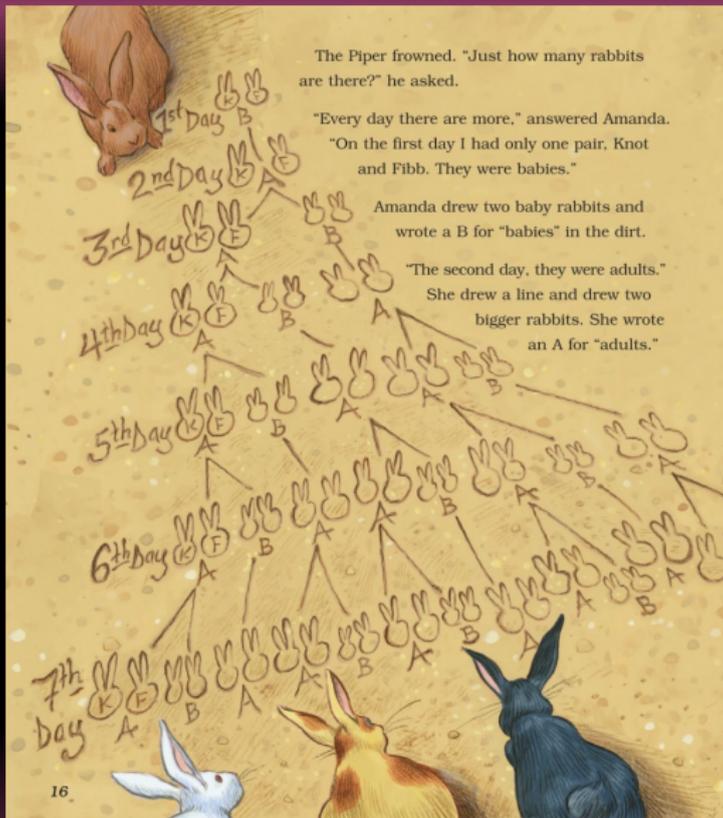
```
In [19]: %timeit F(20)
100 loops, best of 3: 3.4 ms per loop
```

```
In [20]: %timeit F(25)
10 loops, best of 3: 36.7 ms per loop
```

```
In [21]: %timeit F(26)
10 loops, best of 3: 59.1 ms per loop
```

```
In [22]: %timeit F(27)
10 loops, best of 3: 95.9 ms per loop
```

```
In [23]: %timeit F(28)
10 loops, best of 3: 155 ms per loop
```



The Piper frowned. "Just how many rabbits are there?" he asked.

"Every day there are more," answered Amanda.

"On the first day I had only one pair, Knot and Fibb. They were babies."

Amanda drew two baby rabbits and wrote a B for "babies" in the dirt.

"The second day, they were adults."

She drew a line and drew two bigger rabbits. She wrote an A for "adults."

```
suivant ← suivant + courant  
courant ← suivant
```

Exercice 1

À vous de jouer...

```
suivant ← suivant + courant  
courant ← suivant
```

Exercice 1

À vous de jouer...

```
def Fib2(n) :  
    def fibo(a, b, n):  
        if n < 2:  
            return b  
        return fibo(b, a + b, n - 1)  
    return fibo(1, 1, n)
```

```
In [10]: %timeit Fib2(100)  
10000 loops, best of 3: 16.7  $\mu$ s per loop
```

```
def Fib2(n) :  
    def fibo(a, b, n):  
        if n < 2:  
            return b  
        return fibo(b, a + b, n - 1)  
    return fibo(1, 1, n)
```

```
In [10]: %timeit Fib2(100)  
10000 loops, best of 3: 16.7  $\mu$ s per loop
```

Mémoïsation

```
fibcache = {0:1, 1:1}

def Fib3(n):
    if n in fibcache:
        return fibcache[n]
    else:
        fibcache[n] = Fib3(n-1) + Fib3(n-2)
        return fibcache[n]
```

Mémoïsation

```
def Fib4(n) :  
    fibcache = {0:1, 1:1}  
    def Fib_cache(k) :  
        if k in fibcache:  
            return fibcache[k]  
        else:  
            fibcache[k] = Fib_cache(k-1) + Fib_cache(k-2)  
            return fibcache[k]  
    return Fib_cache(n)
```

Mémoïsation

```
from functools import lru_cache

@lru_cache(maxsize = None)
def Fib6(n):
    if n in {0, 1} :
        return 1
    else:
        return Fib6(n - 1) + Fib6(n - 2)
```

Matrices

```
import numpy as np

def Fib7(n) :
    A = np.matrix('0; 1')
    M = np.matrix('0 1; 1 1')
    def fibm(k) :
        if k == 0 :
            return A
        return M * fibm(k-1)
    return fibm(n)[1,0]
```

Matrices

```
In [77]: Fib7(95)
Out[77]: -3659523366270331776
```

??

Matrices

```
In [77]: Fib7(95)
Out[77]: -3659523366270331776
```

??

```
def Fib8(n) :
    M = np.matrix('1 1; 1 0')
    def fibm(k) :
        if k == 0 :
            return M
        return M * fibm(k-1)
    return fibm(n)[1,0]
```

Défi

Pouvez-vous faire mieux ?

```
In [179]: %timeit Fib11(1_000_000)
1 loops, best of 3: 442 ms per loop
```

```
In [180]: %timeit Fib11(2_000_000)
1 loops, best of 3: 1.33 s per loop
```

```
In [181]: %timeit Fib11(4_000_000)
1 loops, best of 3: 4 s per loop
```

```
In [182]: %timeit Fib11(8_000_000)
1 loops, best of 3: 12 s per loop
```

Défi

Pouvez-vous faire mieux ?

```
In [179]: %timeit Fib11(1_000_000)
1 loops, best of 3: 442 ms per loop
```

```
In [180]: %timeit Fib11(2_000_000)
1 loops, best of 3: 1.33 s per loop
```

```
In [181]: %timeit Fib11(4_000_000)
1 loops, best of 3: 4 s per loop
```

```
In [182]: %timeit Fib11(8_000_000)
1 loops, best of 3: 12 s per loop
```

```
In [176]: %time Fib11(9000)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
```

```
Wall time: 861  $\mu$ s
```

```
Out[176]:
```

```
560099116857419990464943760131738213410973574450557569045176277317822939235205
889847949348806304754949278198262172298679574779297161320064939750877571313229
378221575703940626150228439349326844513851927463723955237446597282275096520469
634800231063864188155519959004568780983883593105357198392172284293269487320491
833861266308399371327696698447899888400356574148775600076153447061861327527440
584033178844105541588185668504556952741081485373244865058979409901355133413012
818559073505121293839049046825377352178213218516964697425159896192972280120974
245144761842723070903122738253183331897068595992547070141556418397382375060567
242575538813068672080110283183002890240755687530942784964760570016956150052341
059990556659328616029808803614308992648989681753767230588853838254552288069736
573800942160210638813614154121306378682894407574069653207240176609682089123169
509386337654417370839812009824633889721963803284321598773522232298860383031840
634823339047077980418827154084382134816077040846187007029100525225050411009628
112893973940306163099594091684782251212407706428901440658234672671847511284395
140130231963528229763446863604309308741739055074187039321096756599929637964218
643340665936564726624761980140760894254342275971799732086854574199502772675135
827932516546254945641296103715739922978086817787826932423283473166088163144527
187673348745574685611576129352756991242194682242235344654822077940834629486759
964315191762582992241766782047513771480255336026822877570203431595955451355220
286779101502725771727063258768674327436410642725592705632943547457439584403099
387525913543920938096654576444065751992162827736240140321064859389065030629677
606907206292761054008433597681578671142030125729147906481435883380202596703751
```