Récursivité 1: première approche

MP3 Lycée Montaigne

FIGURE: Mise en abyme : un exemple photographique de récursivité

Plan

"Construisons" la récursivité

Supposons que nous souhaitions fabriquer une fonction affichant la suite des puissances n$^{\text{ième}}$ de 2 dans un ordre décroissant :

"Construisons" la récursivité

Supposons que nous souhaitions fabriquer une fonction affichant la suite des puissances n$^{\text{ième}}$ de 2 dans un ordre décroissant :

facile en faisant appel à une boucle inconditionnelle `for` !!! :...

"Construisons" la récursivité

```
def deux_exp(n):
    for i in range (n+1):
        print 2**(n-i)
deux_exp(5)
```

qui donne :

"Construisons" la récursivité

```
def deux_exp(n):
    for i in range (n+1):
        print 2**(n-i)
deux_exp(5)
```

qui donne :

```
32
16
8
4
2
1
```

"Construisons" la récursivité

```
def deux_exp(n):
    for i in range (n+1):
        print 2**(n-i)
deux_exp(5)
```

qui donne :

```
32
16
8
4
2
1
```

Dans cet exemple, la boucle itérative provoque 6 appels à la fonction `print`.

"Construisons" la récursivité

Nous pourrions limiter le nombre d'itérations en commençant par exemple par afficher 2^5 et en calculant ensuite `deux_exp(4)` :

```
n=5
print 2**n
print deux_exp(n-1)
```

"Construisons" la récursivité

En intégrant ce principe dans une fonction (nouvelle !) cela donne :

```
def deux_exp_2(n):
    print 2**n
    deux_exp(n-1)
```

"Construisons" la récursivité

Ainsi, nous réalisons l'affichage attendu en deux étapes :

"Construisons" la récursivité

Ainsi, nous réalisons l'affichage attendu en deux étapes :

$$\left[\begin{array}{c}\text{l'affichage "direct" de } 2^n \\ \\ \end{array}\right.$$

"Construisons" la récursivité

Ainsi, nous réalisons l'affichage attendu en deux étapes :

$$\left[\begin{array}{c} \text{l'affichage "direct" de } 2^n \\ \text{l'affichage par appel à la fonction de } 2^{n-1},\ 2^{n-2},\ \ldots,\ 2^0 \end{array}\right.$$

"Construisons" la récursivité

Ainsi, nous réalisons l'affichage attendu en deux étapes :

$$\left[\begin{array}{c}\text{l'affichage "direct" de } 2^n \\ \text{l'affichage par appel à la fonction de } 2^{n-1},\ 2^{n-2},\ \ldots,\ 2^0\end{array}\right.$$

- En répétant ce processus, nous pourrions ainsi bâtir autant de fonctions `deux_exp` que "nécessaire" pour parvenir au même résultat.

"Construisons" la récursivité

Ainsi, nous réalisons l'affichage attendu en deux étapes :

$$\left[\begin{array}{c}\text{l'affichage "direct" de } 2^n \\ \text{l'affichage par appel à la fonction de } 2^{n-1},\ 2^{n-2},\ \ldots,\ 2^0\end{array}\right.$$

- En répétant ce processus, nous pourrions ainsi bâtir autant de fonctions `deux_exp` que "nécessaire" pour parvenir au même résultat.
- Inconvénient : il existe alors autant de fonctions que de puissances à calculer, soit $n+1$ finalement et n jamais connu à l'avance ! ! !

CONCLUSION :

"Construisons" la récursivité

Ainsi, nous réalisons l'affichage attendu en deux étapes :

$$\left[\begin{array}{c}\text{l'affichage "direct" de } 2^n \\ \text{l'affichage par appel à la fonction de } 2^{n-1},\ 2^{n-2},\ \ldots,\ 2^0\end{array}\right.$$

- En répétant ce processus, nous pourrions ainsi bâtir autant de fonctions `deux_exp` que "nécessaire" pour parvenir au même résultat.
- Inconvénient : il existe alors autant de fonctions que de puissances à calculer, soit $n+1$ finalement et n jamais connu à l'avance ! ! !

CONCLUSION : USAGE GÉNÉRAL DE CETTE MÉTHODE IMPOSSIBLE !

"Construisons" la récursivité

Itération en programmation impérative

En reprenant l'exemple de calcul de la puissance $n^{\text{ième}}$ de 2, nous pourrions éviter l'appel à la fonction puissance intégrée de Python en exploitant la suite récurrente $u_n = 2 \times u_{n-1}$ de premier terme $u_0 = 1$:

```python
def deux_exp_imp(n):
    res=1
    for i in range (n):
        res=2*res
    return res
print deux_exp_imp(10)
```

La sortie donne :

"Construisons" la récursivité

Itération en programmation impérative

En reprenant l'exemple de calcul de la puissance n$^{\text{ième}}$ de 2, nous pourrions éviter l'appel à la fonction puissance intégrée de Python en exploitant la suite récurrente $u_n = 2 \times u_{n-1}$ de premier terme $u_0 = 1$:

```
def deux_exp_imp(n):
    res=1
    for i in range (n):
        res=2*res
    return res
print deux_exp_imp(10)
```

La sortie donne :

```
1024
```

"Construisons" la récursivité

Passage à la programmation récursive

Il est possible de remplacer ce mode d'évaluation itératif, appelé **programmation impérative**, par une programmation dite **fonctionnelle**, dans laquelle les fonctions s'appellent elles-même. Ce mode de programmation est qualifié de **récursif**. Cela donne pour notre exemple :

```
def deux_exp_rec(n):
    if n==0:
        return 1
    else:
        return 2*deux_exp_rec(n-1)
print deux_exp_rec(5)
```

"Construisons" la récursivité

limitation

le nombre maximum d'appels récursifs est limité à 999 en python.
Que se passe-t-il si l'on lance le script ci-contre ?

```
def deux_exp_rec(n):
    if n==0:
        return 1
    else:
        return 2*deux_exp_rec(n-1)
print deux_exp_rec(999)
```

"Construisons" la récursivité

Exercice

EXERCICE N°1:

Faire tourner le script précédent à la main afin d'écrire les résultats intermédiaires du calcul de `deux_exp_rec(4)`*.*

"Construisons" la récursivité

Exercice

Réponse :

n=4

n-1=3

n-2=2

n-3=1

n-4=0

"Construisons" la récursivité

Exercice

RÉPONSE :

n=4	
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec(3)}}_{=?}$
n-2=2	$\searrow$
n-3=1	
n-4=0	

"Construisons" la récursivité

Exercice

Réponse :

n=4		
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$	
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$
n-3=1		$\searrow$
n-4=0		

"Construisons" la récursivité

Exercice

Réponse :

n=4			
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$		
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$	
n-3=1		$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(1)}_{=?}$
n-4=0			$\searrow$

"Construisons" la récursivité

Exercice

RÉPONSE :

n=4				
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$			
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$		
n-3=1		$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(1)}_{=?}$	
n-4=0			$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(0)}_{=1}$

"Construisons" la récursivité

Exercice

Réponse :

n=4				
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$			
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$		
n-3=1		$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(1)}_{=?}$	$\nwarrow$ 2
n-4=0			$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(0)}_{=1}$

"Construisons" la récursivité

Exercice

Réponse :

n=4				
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$			
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$	$\nwarrow$ 4	
n-3=1		$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(1)}_{=?}$	$\nwarrow$ 2
n-4=0			$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(0)}_{=1}$

"Construisons" la récursivité

Exercice

<u>RÉPONSE :</u>

n=4

n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$	$\nwarrow$ 8		
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$	$\nwarrow$ 4	
n-3=1		$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(1)}_{=?}$	$\nwarrow$ 2
n-4=0			$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(0)}_{=1}$

"Construisons" la récursivité

Exercice

RÉPONSE :

n=4	**16**			
n-1=3	$2 \times \underbrace{\texttt{deux\_exp\_rec}(3)}_{=?}$	$\nwarrow$ 8		
n-2=2	$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(2)}_{=?}$	$\nwarrow$ 4	
n-3=1		$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(1)}_{=?}$	$\nwarrow$ 2
n-4=0			$\searrow$	$2 \times \underbrace{\texttt{deux\_exp\_rec}(0)}_{=1}$

"Construisons" la récursivité

COMMENTAIRES :

"Construisons" la récursivité

COMMENTAIRES :

- les différentes étapes de calcul sont en suspens jusqu'à la dernière **récursion**.

"Construisons" la récursivité

COMMENTAIRES :

- les différentes étapes de calcul sont en suspens jusqu'à la dernière **récursion**.
  $\Longrightarrow$ parfois stockage de nombreux calculs intermédiaires donc forte utilisation de la mémoire $\Longrightarrow$ **forte complexité spatiale**.

"Construisons" la récursivité

COMMENTAIRES :

- les différentes étapes de calcul sont en suspens jusqu'à la dernière **récursion**.
  $\Longrightarrow$ parfois stockage de nombreux calculs intermédiaires donc forte utilisation de la mémoire $\Longrightarrow$ **forte complexité spatiale**.

  La partie de la mémoire dévolue au stockage "intermédiaire" porte le nom de PILE D'EXÉCUTION et est de type "LIFO" pour LAST IN FIRST OUT : le dernier calcul inscrit en mémoire sera évalué le premier, puis l'avant dernier inscrit évalué en second et ainsi de suite jusqu'à la **remontée** au premier appel.

"Construisons" la récursivité

COMMENTAIRES :

- les différentes étapes de calcul sont en suspens jusqu'à la dernière **récursion**.
  $\Longrightarrow$ parfois stockage de nombreux calculs intermédiaires donc forte utilisation de la mémoire $\Longrightarrow$ **forte complexité spatiale**.

  La partie de la mémoire dévolue au stockage "intermédiaire" porte le nom de PILE D'EXÉCUTION et est de type "LIFO" pour LAST IN FIRST OUT : le dernier calcul inscrit en mémoire sera évalué le premier, puis l'avant dernier inscrit évalué en second et ainsi de suite jusqu'à la **remontée** au premier appel.

- Pour la dernière itération, ici $n-4=0$, on rencontre ce que l'on appelle **le cas de base**, structure conditionnelle permettant d'évaluer directement la valeur de la fonction à ce rang final $\Longrightarrow$ le cas de base **assure la terminaison de la fonction**.

"Construisons" la récursivité

Question :

1. Quelle est la hauteur de la pile d'exécution dans l'exemple `deux_exp_rec(5)`.
2. Même question pour le cas général `deux_exp_rec(n)`.

Définition

Définition - (2) - 1:

Une fonction récursive doit contenir les éléments fondamentaux suivants :

- *un (récursivité simple) ou plusieurs (récursivité multiple)* **appel(s) à la fonction elle-même** *lors de son exécution.*
- *un cas de base, c'est à dire* **une situation conditionnelle présente dans la fonction** *qui assure* **sa terminaison**.

Principe de conception d'une fonction récursive

La mise au point d'un algorithme comprenant une procédure récursive de traitement T sur des données D passe par les étapes générales suivantes :

Principe de conception d'une fonction récursive

La mise au point d'un algorithme comprenant une procédure récursive de traitement T sur des données D passe par les étapes générales suivantes :

- Identifier les variables du problème

Principe de conception d'une fonction récursive

La mise au point d'un algorithme comprenant une procédure récursive de traitement T sur des données D passe par les étapes générales suivantes :

- Identifier les variables du problème
- Dégager le cas de base qui doit conduire **à la terminaison de la procédure récursive**.

Principe de conception d'une fonction récursive

La mise au point d'un algorithme comprenant une procédure récursive de traitement T sur des données D passe par les étapes générales suivantes :

- Identifier les variables du problème
- Dégager le cas de base qui doit conduire **à la terminaison de la procédure récursive**.
- Décomposer le traitement T en un ensemble de sous-traitements identiques au traitement de départ et dans lequel les variables convergent vers le cas de base.

Principe de conception d'une fonction récursive

La mise au point d'un algorithme comprenant une procédure récursive de traitement T sur des données D passe par les étapes générales suivantes :

- Identifier les variables du problème
- Dégager le cas de base qui doit conduire **à la terminaison de la procédure récursive**.
- Décomposer le traitement T en un ensemble de sous-traitements identiques au traitement de départ et dans lequel les variables convergent vers le cas de base.
- Ecrire l'algorithme.

Quelques exemples classiques simples

Factorielle

Un cas ultra classique d'usage de la récursivité est le calcul de la fonction factorielle, donc le script Python est le suivant :

```
def fact(N):
    if N==1: #cas de base!!!
        return 1
    else:
        return N*(fact(N-1)) #rÃ©currence convergent vers le cas de base
n=int(input("Entrez un entier positif n: "))
print fact(n)
```

Quelques exemples classiques simples

PGCD récursif

L'un des algorithmes itératifs de calcul du PGCD de deux nombres a et b s'appuie sur la division euclidienne (algorithme d'Euclide). Son principe est le suivant :

Quelques exemples classiques simples

PGCD récursif

L'un des algorithmes itératifs de calcul du PGCD de deux nombres a et b s'appuie sur la division euclidienne (algorithme d'Euclide). Son principe est le suivant :

- on calcule le reste de $a//b$ que l'on stocke dans r.

Quelques exemples classiques simples

PGCD récursif

L'un des algorithmes itératifs de calcul du PGCD de deux nombres a et b s'appuie sur la division euclidienne (algorithme d'Euclide). Son principe est le suivant :

- on calcule le reste de $a//b$ que l'on stocke dans r.
- tant que $a\%b \neq 0$ faire : $\begin{cases} r = a\%b \\ a = b \\ b = r \end{cases}$

Quelques exemples classiques simples

PGCD récursif

L'un des algorithmes itératifs de calcul du PGCD de deux nombres a et b s'appuie sur la division euclidienne (algorithme d'Euclide). Son principe est le suivant :

- on calcule le reste de $a//b$ que l'on stocke dans r.
- tant que $a\%b \neq 0$ faire : $\begin{cases} r = a\%b \\ a = b \\ b = r \end{cases}$
- On renvoie b.

Quelques exemples classiques simples

PGCD récursif

Un exemple de rotation "à la main" est le suivant : cherchons le PGCD de 96 et 81, soit

a	b	r

Quelques exemples classiques simples

PGCD récursif

Un exemple de rotation "à la main" est le suivant : cherchons le PGCD de 96 et 81, soit

$$\begin{array}{ccccc} a & & b & & r \\ 96 & = & 1 \times 81 & + & 15 \end{array}$$

Quelques exemples classiques simples

PGCD récursif

Un exemple de rotation "à la main" est le suivant : cherchons le PGCD de 96 et 81, soit

$$\begin{array}{ccccc} a & & b & & r \\ 96 & = & 1 \times 81 & + & 15 \\ 81 & = & 5 \times 15 & + & 6 \end{array}$$

Quelques exemples classiques simples

PGCD récursif

Un exemple de rotation "à la main" est le suivant : cherchons le PGCD de 96 et 81, soit

$$
\begin{array}{cccccc}
a & & b & & r & \\
96 & = & 1 \times 81 & + & 15 & \\
81 & = & 5 \times 15 & + & 6 & \\
15 & = & 2 \times 6 & + & 3 & \text{la boucle s'arrête ici car.....}
\end{array}
$$

Quelques exemples classiques simples

PGCD récursif

Un exemple de rotation "à la main" est le suivant : cherchons le PGCD de 96 et 81, soit

$$\begin{array}{ccrcc}
a & & b & & r \\
96 & = & 1 \times 81 & + & 15 \\
81 & = & 5 \times 15 & + & 6 \\
15 & = & 2 \times 6 & + & 3 \\
6 & = & 2 \times \boxed{3} & + & 0
\end{array}$$

la boucle s'arrête ici car.....
sa condition est violée au rang suivant !

Quelques exemples classiques simples

PGCD récursif

Exercice n°2:

Proposer un algorithme récursif de calcul du PGCD.

Quelques exemples classiques simples

PGCD récursif

```python
def pgcdrec(a,b):
    if a<=0 or b<=0:
        raise ValueError()
    if (a%b)==0:
        return b #cas de base
    else:
        return pgcdrec(b,a%b)

na=int(input("Entrez le premier nombre: "))
nb=int(input("Entrez le second nombre: "))
try:
    res=pgcdrec(na,nb)
    print(res)
except ValueError:
        print("Parametre_negatif ou nul")
```

Quelques exemples classiques simples

Conjecture de Syracuse

EXERCICE N°3:

On définit la suite de Syracuse par :

$$\left[\begin{array}{ccc} x_1 & = & a \in \mathbb{N}^* \\ x_{n+1} & = & \begin{cases} \dfrac{x_n}{2} & si\ x_n\ est\ pair \\ 3 \times x_n + 1 & si\ x_n\ est\ impair \end{cases} \end{array}\right.$$

1. Proposer une fonction Python `SyracuseRec(a,n)` qui calcule de manière récursive le $n^{\text{ième}}$ terme d'une suite de Syracuse de premier terme a.
2. Réaliser un script de programme principal exploitant la fonction `SyracuseRec(a,n)`, et permettant l'affichage des 8 termes qui suivent celui obtenu à un certain rang lorsqu'il vaut 1. Conclure sur la structure de la suite des nombres suivants.

Quelques exemples classiques simples

Conjecture de Syracuse : réponse du 1

RÉPONSE :

```
def Syracuse(a,n):
    if n==1:
        return a
    else:
        if (Syracuse(a,n-1)%2)==0:
            return Syracuse(a,n-1)//2
        else:
            return 3*Syracuse(a,n-1)+1
print Syracuse(1,6)
```

Quelques exemples classiques simples

Conjecture de Syracuse : réponse du 2

```
a=0.0
n=1
while not(type(a)==int) and not(a>0):
    a=input(u"Entrez la valeur de a entier positif:")

res=0
n=1
liste=[]
while res!=1:
    res=Syracuse(a,n)
    print res
    n=n+1
for k in range(n-1,n+8):
   liste.append(Syracuse(a,k))
print liste
```

La sortie donne :

Entrez la valeur de a entier positif :1

$$1, 4, 2, 1, 4, 2, 1, 4, 2$$

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme naïf permettant le calcul de n^p, sans faire appel à la fonction puissance intégrée de Python, consiste à multiplier n par lui-même p fois. Cette manière de faire conduit à une complexité «en gros» $\mathcal{O}(p)$. Il est possible d'améliorer sensiblement le calcul en exploitant un algorithme récursif dit **d'exponentiation rapide**.

Quelques exemples classiques simples

Exponentiation rapide

Quelques notes sur la complexité :

L'exposant p peut toujours être décomposé en base 2 avec [1] :

$$p = \sum_{i=0}^{d} b_i \times 2^i \qquad \text{avec } b_i = \{0,1\}$$

on a alors : $n^p = n^{\sum_{i=0}^{d} b_i \times 2^i} = n^{b_0}(n^2)^{b_1}(n^{2^2})^{b_2}\dots(n^{2^d})^{b_d}$
Il faudra $d+1$ opérations pour calculer les $(n^{2^i})^{b_i}$, puis encore d opérations pour former leur produit. Ainsi, il faut $2d+1$ opérations au total,donc une complexité «en gros» de $\mathcal{O}(d)$. La complexité se calcule facilement :

$$\ln p \sim ln(2^d) \;\Rightarrow\; C \sim d \sim \frac{\ln p}{\ln 2} = \log_2 p < p$$

1. cf cours révisions 3 "Représentation des nombres en machine"

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme est le suivant pour tout entier $n > 1$:

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme est le suivant pour tout entier $n > 1$:

- si n est pair alors $x^n = (x^2)^{\frac{n}{2}}$; on calcule alors $y^{\frac{n}{2}}$ avec $y = x^2$

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme est le suivant pour tout entier $n > 1$:

- si n est pair alors $x^n = (x^2)^{\frac{n}{2}}$; on calcule alors $y^{\frac{n}{2}}$ avec $y = x^2$
- si n est impair alors $x^n = x \times (x^2)^{\frac{n-1}{2}}$; on calcule alors $y^{\frac{n-1}{2}}$ avec $y = x^2$ que l'on multiplie ensuite par x

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme est le suivant pour tout entier $n > 1$:

- si n est pair alors $x^n = (x^2)^{\frac{n}{2}}$; on calcule alors $y^{\frac{n}{2}}$ avec $y = x^2$
- si n est impair alors $x^n = x \times (x^2)^{\frac{n-1}{2}}$; on calcule alors $y^{\frac{n-1}{2}}$ avec $y = x^2$ que l'on multiplie ensuite par x

En résumé, tout cela devient :

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme est le suivant pour tout entier $n > 1$:

- si n est pair alors $x^n = (x^2)^{\frac{n}{2}}$; on calcule alors $y^{\frac{n}{2}}$ avec $y = x^2$
- si n est impair alors $x^n = x \times (x^2)^{\frac{n-1}{2}}$; on calcule alors $y^{\frac{n-1}{2}}$ avec $y = x^2$ que l'on multiplie ensuite par x

En résumé, tout cela devient :

$$puissance(x,n) = \left[\begin{array}{cc} x & \text{si } n = 1 \\ puissance(x^2, n/2) & \text{si } n \text{ est pair} \\ x \times puissance(x^2, n//2) & \text{si } n > 2 \text{ est impair} \end{array} \right.$$

Quelques exemples classiques simples

Exponentiation rapide

L'algorithme est le suivant pour tout entier $n > 1$:

- si n est pair alors $x^n = (x^2)^{\frac{n}{2}}$; on calcule alors $y^{\frac{n}{2}}$ avec $y = x^2$
- si n est impair alors $x^n = x \times (x^2)^{\frac{n-1}{2}}$; on calcule alors $y^{\frac{n-1}{2}}$ avec $y = x^2$ que l'on multiplie ensuite par x

En résumé, tout cela devient :

$$puissance(x, n) = \left[\begin{array}{cc} x & \text{si } n = 1 \\ puissance(x^2, n/2) & \text{si } n \text{ est pair} \\ x \times puissance(x^2, n//2) & \text{si } n > 2 \text{ est impair} \end{array}\right.$$

Exercice n°9:

Ecrire le script récursif en Python de la fonction d'exponentiation rapide.

Quelques exemples classiques simples

Exponentiation rapide

<u>RÉPONSE :</u>

```
import time as t
def exporapide(x,n):
    if n==1:
        return x
    elif (n%2)==0:
        return exporapide(x**2,n/2)
    else:
        return x*exporapide(x**2,n//2)

for k in range(1,100):
    print (u"exposant␣de␣2:"),k
    a=t.clock()
    print(u"RÃ©sultat:"),exporapide(2,k)
    b=t.clock()
    print(u"temps␣d'exÃ©cution␣en␣rÃ©cursif␣:"), b-a
```

Quelques exemples classiques simples

Exponentiation rapide : résultat

```
exposant de 2 : 12
Résultat : 4096
temps d'exécution en récursif : 9.1107228717e-05
exposant de 2 : 13
Résultat : 8192
temps d'exécution en récursif : 0.000193763261074
exposant de 2 : 14
Résultat : 16384
temps d'exécution en récursif : 0.000159758450356
exposant de 2 : 15
Résultat : 32768
temps d'exécution en récursif : 0.0126202759779
```

Les "dangers" de la récursivité

Parfois, les méthodes récursives engendrent des temps de calculs bien trop longs en raison de trop nombreux appels à la fonction $\Longrightarrow$ **méthode itérative plus performante ! ! !**

Les "dangers" de la récursivité

Suite de Fibonacci

EXEMPLE : **suite de Fibonacci** démarrant à $u_0 = 0$ puis $u_1 = 1$

```
def fiborec(n):
    #algorithme rÃ©cursif
    if n==1 or n==2 :
        return 1
    return fiborec(n-1)+fiborec(n-2)

for k in range(5):
    print((k+1)*10)
    a=time.clock()
    fiborec((k+1)*10)
    b=time.clock()
    print("temps␣d'exÃ©cution␣en␣rÃ©cursif␣:", b-a
      )
```

```
rang : 10
temps d'exécution en récursif : 2.95136093027e-05
rang : 20
temps d'exécution en récursif : 0.00322596581682
rang : 30
temps d'exécution en récursif : 0.449604549715
rang : 40
temps d'exécution en récursif : 53.8201081353
```

```
import time as t
def fiboiter(n):
    # algorithme itÃ©ratif
    i,j,k,s=1,1,3,2
    if n==1 or n==2 :
        return 1
    else:
        while k<=n :
            s=i+j
            i=j
            j=s
            k+=1
    return s

```

```
rang : 10
temps d'exécution en itératif : 4.49120141563e-06
rang : 20
temps d'exécution en itératif : 4.49120141563e-06
rang : 30
temps d'exécution en itératif : 5.77440182009e-06
rang : 40
temps d'exécution en itératif : 7.05760222456e-06
```

Récursivité simple

Définition - (1) - 2:

La récursivité simple correspond au cas le plus classique pour lequel la fonction récursive ne fait qu'un seul appel à elle-même lors de chaque récurrence

On peut citer parmi les algorithme déjà vus : le PGCD récursif, factorielle, suite de Syracuse, exponentiation rapide etc..

Récursivité multiple

DÉFINITION - (2) - 3:

La récursivité multiple correspond au cas d'une fonction récursive faisant plus d'un appel à elle-même lors de chaque récurrence.

NB : la suite de Fibonacci, traitée plus haut, est déjà un exemple de récursivité multiple.

Récursivité multiple

Un bon exemple de récursivité multiple est le calcul des coefficients binomiaux $C_n^p = \binom{n}{p} = \dfrac{n!}{p!(n-p)!}$ avec $(n,p) \geq (0,0)$
Ainsi on a : $C_0^0 = C_n^n = 1$. En outre on montre facilement la formule de Pascal qui lie les coefficients binomiaux :

$$C_{n+1}^{p+1} = C_n^p + C_n^{p+1}$$

à l'aide du triangle de Pascal, que l'on fabrique en plaçant des 1 sur les extrémités de chaque ligne indicée n puisque $C_0^0 = C_n^n = 1$, les termes inférieurs s'obtenant par sommation des termes adjacents de la ligne supérieure.

$$\begin{array}{c|ccccccccccccc}
n=0 & & & & & & & 1 & & & & & & \\
n=1 & & & & & & 1 & & 1 & & & & & \\
n=2 & & & & & 1 & & 2 & & 1 & & & & \\
n=3 & & & & 1 & & 3 & & 3 & & 1 & & & \\
n=4 & & & 1 & & 4 & & 6 & & 4 & & 1 & & \\
n=5 & & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & \\
n=6 & 1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1
\end{array}$$

Récursivité multiple

Cette récurrence permet de facilement calculer les coefficients binomiaux C_n^p par une fonction récursive multiple :

```
def C(n,p):
    if (n>p) and (p>0):
        return C(n-1,p-1)+C(n-1,p)
    else: #cas de base
        return 1
```

Récursivité imbriquée

Définition - (3) - 4:

Un fonction récursive dont **l'un des paramètres est un appel à elle-même** *est qualifiée de* **fonction récursive imbriquée**

Récursivité imbriquée

Exemples

Exercice n°10:

On donne les fonctions d'Ackermann $A(m,n)$ et de Morris $M(m,n)$ définies par :

$$A(m,n) = \left[\begin{array}{cl} n+1 & si\ m=0\ et\ n\geq 1 \\ A(m-1,1) & si\ n=0\ et\ m\geq 1 \\ A(m-1,A(m,n-1)) & si\ n\geq 1\ et\ m\geq 1 \end{array} \right.$$

$$M(m,n) = \left[\begin{array}{cc} 1 & si\ m=0 \\ M(m-1,M(m,n)) & si\ n\geq 1\ et\ m\geq 1 \end{array} \right.$$

Ecrire les scripts récursifs Python de ces deux fonctions.

Récursivité imbriquée

Exemples : réponses

RÉPONSES :

```
def A(m,n):
    if (m==0) and (n>=1): # cas de base
        return n+1
    elif (n==0) and (m>=1):
        return A(m-1,1)
    else:
        return A(m-1,A(m,n-1))
```

```
def M(m,n):
    if (m==0): # cas de base
        return 1
    else:
        return M(m-1,M(m,n))
```

Récursivité croisée

Définition - (4) - 5:

Deux fonctions récursives sont dites **mutuellement récursives** *lorsqu'elles s'appellent l'une l'autre. On parle alors* **récursivité croisée**.

Récursivité croisée

Premier exemple

■ <u>EXEMPLE :</u>

Un exemple très classique de récursivité croisée est l'évaluation de la parité d'un nombre :

```
def pair(n):
if n==0:
return True
else:
return impair (n-1)
```

```
def impair(n):
if n==0:
return False
else:
return pair(n-1)
```

Récursivité croisée

Premier exemple

Question : Quel est le résultat de l'appel `pair(2n+1)` ?

Réponse :

Récursivité croisée

Premier exemple

QUESTION : Quel est le résultat de l'appel `pair(2n+1)` ?

RÉPONSE :

$2n+1$ `pair(2n+1)`

Récursivité croisée

Premier exemple

<u>Question :</u> Quel est le résultat de l'appel `pair(2n+1)` ?

<u>Réponse :</u>

$2n+1$ $\quad$ $\mathtt{pair}(2\mathtt{n}+1)$

$(2n+1)-1$ $\quad$ $\mathtt{impair}((2\mathtt{n}+1)-1)$

Récursivité croisée

Premier exemple

QUESTION : Quel est le résultat de l'appel `pair(2n+1)` ?

RÉPONSE :

$2n+1$	$\texttt{pair}(2n+1)$		
$(2n+1)-1$		$\texttt{impair}((2n+1)-1)$	
$(2n+1)-2$			$\texttt{pair}((2n+1)-2)$

Récursivité croisée

Premier exemple

QUESTION : Quel est le résultat de l'appel pair(2n+1) ?

RÉPONSE :

$2n+1$	$\mathtt{pair}(2n+1)$			
$(2n+1)-1$		$\mathtt{impair}((2n+1)-1)$		
$(2n+1)-2$			$\mathtt{pair}((2n+1)-2)$	
$(2n+1)-3$				$\mathtt{impair}((2n+1)-3)$

Récursivité croisée

Premier exemple

QUESTION : Quel est le résultat de l'appel `pair(2n+1)` ?

RÉPONSE :

$2n+1$	$\mathtt{pair}(2n+1)$				
$(2n+1)-1$		$\mathtt{impair}((2n+1)-1)$			
$(2n+1)-2$			$\mathtt{pair}((2n+1)-2)$		
$(2n+1)-3$				$\mathtt{impair}((2n+1)-3)$	
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Récursivité croisée

Premier exemple

QUESTION : Quel est le résultat de l'appel `pair(2n+1)` ?

RÉPONSE :

$2n+1$	$\texttt{pair}(2\texttt{n}+1)$				
$(2n+1)-1$		$\texttt{impair}((2\texttt{n}+1)-1)$			
$(2n+1)-2$			$\texttt{pair}((2\texttt{n}+1)-2)$		
$(2n+1)-3$				$\texttt{impair}((2\texttt{n}+1)-3)$	
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$2n+1-(2n)=1$					$\texttt{pair}(1)$

Récursivité croisée

Premier exemple

QUESTION : Quel est le résultat de l'appel pair(2n+1) ?

RÉPONSE :

$2n+1$	$\mathtt{pair}(2\mathtt{n}+1)$					
$(2n+1)-1$		$\mathtt{impair}((2\mathtt{n}+1)-1)$				
$(2n+1)-2$			$\mathtt{pair}((2\mathtt{n}+1)-2)$			
$(2n+1)-3$				$\mathtt{impair}((2\mathtt{n}+1)-3)$		
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	
$2n+1-(2n)=1$					$\mathtt{pair}(1)$	
$2n+1-(2n+1)=0$						$\mathtt{impair}(0)$

Récursivité croisée

Premier exemple

<u>QUESTION :</u> Quel est le résultat de l'appel `pair(2n+1)` ?

<u>RÉPONSE :</u>

$$\begin{array}{ccccccc}
2n+1 & \mathtt{pair}(2n+1) & & & & & \\
(2n+1)-1 & & \mathtt{impair}((2n+1)-1) & & & & \\
(2n+1)-2 & & & \mathtt{pair}((2n+1)-2) & & & \\
(2n+1)-3 & & & & \mathtt{impair}((2n+1)-3) & & \\
\dots & \dots & \dots & \dots & \dots & \dots & \\
2n+1-(2n)=1 & & & & & \mathtt{pair}(1) & \\
2n+1-(2n+1)=0 & & & & & & \mathtt{impair}(0)
\end{array}$$

<u>CONCLUSION :</u> le résultat est évidemment le booléen `False` ! ! ! !