Récursivité et récurrence

F. Didier

18 décembre 2015

Résumé

Les termes *récursif*, *récursion*, *récursivité* ont la même racine que le mot *récurrence*. Le mot latin *currere* signifiant courir et par là *recurrere* signifie courir en arriere.
Le terme récurrence est apparu au début du 20è siècle. On parle alors de formules de récurrence et de raisonnement par récurrence pour parler du raisonnement par induction introduit par Blaise Pascal.
La récursivité apparaît à la fin des années soixante avec le premier langage de programmation, Algol 60, qui permet l'écriture de programmmes récursifs. Cela explique pourquoi le terme récurrence est utilisé principalement en mathématiques alors que le terme récursivité est utilisé par les informaticiens. Les deux notions permettent de généraliser des propriétés, mais une, le raisonnement par récurrence, part des cas triviaux pour ensuite généraliser, alors que l'autre, le raisonnement récursif, part du général pour se ramener aux cas triviaux. Mais la récursivité est aussi utilisée en mathématiques comme on va le constater sur quelques exemples.
L'algorithme de dérivation d'une expression qui est enseigné au lycée est un algorithme purement récursif. On n'en a pas souvent conscience. En effet on enseigne les règles à appliquer suivant l'opérateur considéré : la dérivée d'une somme $u + v$ est la somme des dérivées de chacun des termes $u' + v'$, la règle pour la dérivée d'un produit, pour la dérivée d'un quotient, etc... et enfin les règles terminales pour certaines expressions élémentaires : constantes, monômes, fonctions ,...qui permettent d'arrêter les imbrications. Lorsqu'on dérive une expression on utilise un algorithme récursif. Heureusement que les expressions que l'on dérive au lycée sont peu complexes, ce qui permet en général d'obtenir le résultat recherché. Il n'est pas étonnant que l'algorihme de dérivation soit récursif car les données qu'il manipule, à savoir les expressions, sont elles mêmes définies récursivement. Schématiquement, une expression consiste en des termes séparés par des opérateurs additifs, chaque terme consiste en des facteurs séparés par des opérateurs multiplicatifs,..., pour aboutir à des expressions primaires comme : variables,

constantes, (expression),... Ainsi dans la définition de la syntaxe d'une expression on retrouve une référence à expression. C'est la définition même de la notion de récursivité. Plus généralement, on peut dire que lorsque des données sont définies récursivement, naturellement, les algorithmes qui les manipulent sont des algoritmes récursifs. C'est les cas pour tous les algorithmes qui opèrent sur des structures de données très importantes en informatique comme les listes, les arbres, ...

Il existe d'autres domaines en mathématiques où la récursivité intervient. Les résultats d'analyse combinatoire présentés aux élèves de second cycle de l'enseignement secondaire portent sur les dénombrements de certains ensembles finis : Sous-ensembles, injections, bijections, parties à p éléments d'un ensemble fini de cardinalité n,....
Toutes les démonstrations portent sur la mise en évidence d'un procédé récurrent d'énumération des ensembles que l'on dénombre. Il apparaît trop souvent que l'on ne retient que les formules combinatoires obtenues en oubliant les procédés utilisés. Les systèmes d'écriture couramment utilisés dans ce domaine permettent bien d'écrire des formules statiques (qui deviennent alors très souvent des définitions). Par exemple, on retient que le nombre de bijections est $n!$, que le nombre d'injections $\frac{n!}{(n-p)!}$, que le nombre de combinaisons est $\frac{n!}{p!(n-p)!}$, ...en oubliant le raisonnement (très souvent récursif) qui a permis de les établir. Dans les cas évoqués, l'écriture des formules statiques n'est rendue possible que parce qu'un symbolisme particulier a été introduit pour noter la fonction factorielle. Ces écritures sont considérées, en mathématiques, plus adaptées pour obtenir une évaluation qu'une écriture récursive. Le défaut principal de ces écritures est qu'elles ne mettent pas en évidence des algorithmes de calcul ou d'énumération.

C'est la nature du procédé d'énumération qui est fondamentale, le dénombrement de ces objets n'étant qu'une conséquence de celui-ci. Le type de raisonnement qui est mis en oeuvre est ce que les informticiens appellent raisonnement "récursif" : il est pour reprendre l'expression de Blaise PASCAL (qui est semble-t-il le premier à l'avoir utilisé) "de nature universelle et porte sa demonstration en soi". Aujourd'hui, avec banalisation de l'outil informatique les définitions récursives de ces dénombrements permettent une évaluation tout aussi facile que celle que l'on obtient en demandant l'évaluation d'une formule statique. Au contraire, comme dans le cas du dénombrement des surjections, Il est plus aisé d'évaluer l'écriture récursive que l'écriture statique.

Ce document présente des exemples d'algorithmes récursifs en lien étroit avec les programmes de mathématiques du lycée. Ces exemples peuvent venir en complément des grands classiques : tours de Hanoï, quicksort, recherche dichotomique,... Il permet d'avoir une réflexion sur l'influence de l'informatique sur l' enseignement des mathématiques.

1 Enumeration et dénombrement des combinaisons

Soient n et p deux entiers, E un ensemble à n éléments, nous noterons $\mathfrak{P}(p,n)$ l'ensemble des sous-ensembles de E ayant p éléments et C_n^p son cardinal.
il est clair que

$$\mathfrak{P}(p,E)=\emptyset, \text{ si } p > n$$

et donc nous avons :

$$C_n^p = 0, \text{ si } p > n$$

De plus, si $p = 0$ il est non moins évident que :

$$\mathfrak{P}(p,E)=\{\emptyset\}$$

et donc que :

$$C_n^p = 1$$

Ces cas singuliers étudiés, envisageons maintenant la cas où $1 \leq p \leq n$. Soit e un élément de E que nous particularisons. Enumérer les parties de E à p éléments peut se faire en énumérant les parties à p éléments de $E \setminus \{e\}$, et en énumérant ensuite les parties de E à p éléments qui contiennent e (ce qui revient, en fait, à énumérer les parties à $p-1$ éléments de $E \setminus \{e\}$. Compte tenue des notations introduites cette énumération repose sur la partition suivante de $\mathfrak{P}(p,E)$:

$$\mathfrak{P}(p,E)=\mathfrak{P}_1 \bigcup \mathfrak{P}_2$$

où

$$\mathfrak{P}_1=\mathfrak{P}(p,E-\{e\})$$

et $\mathfrak{P}_2$ est l'ensemble des parties de la forme $\{e\} \bigcup P$, P décrivant l'ensemble

$$\mathfrak{P}(p-1,E-\{e\})$$

d'où

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

Si nous regroupons les résultats obtenus, il est alors possible d'affirmer que $\mathfrak{P}(p,E)$ et C_n^p sont entièrement déterminés par les formules :

$\mathfrak{P}(p,E)$=
si $p = 0$ **alors**
$\{\phi\}$
sinon
si $p > n$ **alors**
ϕ
sinon
$\mathfrak{P}_1 \bigcup \mathfrak{P}_2$

et

$C_n^p =$
si $p = 0$ **alors**
1
sinon
si $p > n$ **alors**
0
sinon
$C_{n-1}^p + C_{n-1}^{p-1}$

1.1 Programme Python

```
# L'ensemble des combinaisons sera représenté par une liste d'ensembles.
# Le principe est le suivant:
# On choisit aléatoirement un élément e de E, l'ensemble des combinaisons est partitionné en
# les combinaisons à p éléments qui contiennent e et celles à p éléments
# qui ne contiennent pas e. Pour les premières il suffit d'ajouter l'élément e
# à chacune des combinaisons à p-1 éléments de E-{e}.

def choisir_elmt(E):
    # Pour choisir aléatoirement un élément de l'ensemble E
    # On doit convertir E en une liste
    lst=list(E)
    i=int(len(E)*random())
    return lst[i]

def ajout (elmt,lst_ens): # ajoute elmt à chacun des ensembles de la liste
    if len (lst_ens)==0:
        return [set(elmt)]
    else:
        res=[]
        for x in lst_ens:
            res= res + [set(elmt)|x]
        return res

                # énumération des parties à p éléments de E
def comb(E,p):
    if p==0:
        return [set([])]
    elif p>len(E):
```

```
        return []
    else:
        e=choisir_elmt(E)
        return ajout (e,comb(E-set(e),p-1))+ comb(E-set(e),p)

        # dénombrement des parties à p éléments d'un ensemble ayant n éléments

def C(n,p):
    if p==0:
        return 1
    elif p>n:
        return 0
    else:
        return C(n-1,p-1)+C(n-1,p)

#Programme Principal

from random import*
E=set(raw_input("Donner un ensemble de lettres : "))
p= int(raw_input("Donner p : "))

# Enumération
print
print "Enumération des parties à" ,p," éléments de",list(E)
print comb(E,p)

#Dénombrement
n=len(E)
print
print "Nombres de combinaisons : "
print C(n,p)
```

Voici le résultat d'une exécution de ce programme dans le cas où l'on veut énumérer et compter le nombre de parties à deux éléments d'un ensemble de quatre lettres a,b,c,d

```
Donner un ensemble de lettres : abcd
Donner p : 2

Enumération des parties à 2  éléments de ['a', 'c', 'b', 'd']
[set(['a', 'd']), set(['a', 'b']), set(['a', 'c']), set(['c', 'b']),
 set(['b', 'd']), set(['c', 'd'])]

Nombres de combinaisons:
6
```

On peut sophistiquer l'affichage des ensembles pour avoir une représentation analogue à celle généralement utilisée en mathématiques : les éléments de l'ensemble sont écrits entre une paire d'accolades. Pour cela on pourra utiliser les deux procédures ci-dessous et remplacer dans le programme principal l'instruction $printcomb(E)$ par $affiche\_list\_ens(comb(E))$

```
def affiche_ens (E):
    print "{",
    lst=list(E)
```

```
    if len(E)>1:
        for i in xrange(0,len(lst)-1):
            print lst[i],",",
    print lst[len(lst)-1],
    print"}",

def affiche_list_ens (L):#Affiche une liste d'ensembles
    print "[",
    for i in xrange(0,len(L)-1):
        affiche_ens (L[i])
        print",",
    affiche_ens(L[len(L)-1])
    print"]"
```

Voici le résultat produit par cet affichage :

```
Donner un ensemble de lettres : abcd
Donner p : 2
Enumération des parties à 2  éléments de ['a', 'c', 'b', 'd']
[ { b , d } , { c , b } , { a , b } , { c , d } , { a , c } , { a , c
Nombre de combinaisons :  6
```

2 Enumération et dénombrement des parties

Soit n un entier, E un ensemble à n éléments, nous noterons $\mathfrak{P}(n)$ l'ensemble des parties de E. il est clair que

$$\mathfrak{P}(E)=\{\phi\}, \text{ si } n = 0$$

Ce cas singulier étudié, envisageons maintenant la cas où $n \neq 0$. Soit e un élément de E que nous particularisons. Enumérer les parties de E éléments peut se faire en énumérant les parties de $E \setminus \{e\}$, et en énumérant ensuite les parties de E qui contiennent e (ce qui revient, en fait, à ajouter e à chacune des parties de $E \setminus \{e\}$). Compte tenu des notations introduites cette énumération repose sur la partition suivante de $\mathfrak{P}(E)$:

$$\mathfrak{P}(E)=\mathfrak{P}_1 \bigcup \mathfrak{P}_2$$

où

$$\mathfrak{P}_1=\mathfrak{P}(E - \{e\})$$

et $\mathfrak{P}_2$ est l'ensemble des parties de la forme $\{e\} \bigcup P$, P décrivant l'ensemble

$$\mathfrak{P}(E - \{e\})$$

d'où

$$P_n = 2 * P_{n-1}.$$

Si nous regroupons les résultats obtenus, il est alors possible d'affirmer que $\mathfrak{P}(E)$ et P_n sont entièrement déterminés par les formules :

$\mathfrak{P}(E)$=
si $p = 0$ **alors**
$\{\phi\}$
sinon
$\mathfrak{P}_1 \bigcup \mathfrak{P}_2$

et

P_n=
si $p = 0$ **alors**
1
sinon
$2 * P_{n-1}$

2.1 Programme Python

```
# On représente l'ensemble des parties d'un ensemble E par une liste d'ensembles.
# Le principe est le suivant:
# On choisit aléatoirement un élément e. L'ensemble des parties de E est constitué de celles
# qui contiennent e et de celles qui ne contiennent pas e.
# C'est une partition de l'ensemble des parties.
# Ainsi il suffit de faire l'union entre Parties (E-{e}) et {e} + Parties((E-{e}))
# où + signifie que l'on ajoute l'élément e à chacune des parties de E-{e}.

def choisir_elmt(E):  # choix aléatoire d'un élément de l'ensemble e
    lst=list(E)
    i=int(len(E)*random())
    return lst[i]

def ajout (elmt,lst_ens): # ajoute elmt à chacun des ensembles de la liste
    if len (lst_ens)==0:
        return [set(elmt)]
    else:
        res=[]
        for x in lst_ens:
            res= res + [set(elmt)|x]
        return res


def parties(E):
    if len(E)==0:
        return [set([])]
    else:
        e=choisir_elmt(E)
        return ajout (e,parties(E-set(e)))+ parties(E-set(e))
```

```
def nb_parties(n):
    if n==0:
        return 1
    else:
        return 2*nb_parties(n-1)

#      **** Procédures pour l'affichage ****

def affiche_ens (E):
    print "{",
    lst=list(E)
    if len(E)>1:
        for i in xrange(0,len(lst)-1):
            print lst[i],",",
    if len (lst)>0:
            print lst[len(lst)-1],
    print"}",

def affiche_list_ens (L):#Affiche une liste d'ensembles
    print "[",
    for i in xrange(0,len(L)-1):
        affiche_ens (L[i])
        print",",
    affiche_ens(L[len(L)-1])
    print"]"

#Programme Principal

from random import*
E=set(raw_input("Donner un ensemble de lettres : "))
affiche_list_ens parties(E)
print "Nombre de parties: ", nb_parties(len(E))
```

Voici le résultat d'une exécution de ce programme dans le cas où l'on veut énumérer et compter le nombre de parties d'un ensemble de quatre lettres a,b,c,d

```
Donner un ensemble de lettres : abcd
[ { a , c , b , d } , { a , b , d } , { a , c , b } , { a , b } , { a
Nombre de parties:  16
```

3 Enumération et dénombrement des bijections

Soit E et F deux ensembles finis de cardinal n, et notons $\mathfrak{B}(E,F)$ l'ensemble des applications bijectives de E dans F et $B(n)$ son cardinal.
Si $n = 0$, nous avons :

$$\mathfrak{B}(E,F)=\emptyset,$$

et donc :

$$B(0) = 1$$

Examinons maintenant le cas où $n > 0$. Soit e un élément de E que nous particularisons. g est une bijection de E sur F, si f est l'image de e par g (alors $g(e) = f$), et si nous enlevons le couple (e, f) du graphe de g, nous obtenons le graphe d'une bijection de $E - \{e\}$ sur $F - \{f\}$. Réciproquement si $f \in F$ et si $\bar{f}$ est une bijection de $E - \{e\}$ dans $F - \{f\}$, il existe une seule bijection de E dans F, disons $\bar{g}$, qui verifie :
$g(e) = f$ et ($\forall x \in E - \{e\}$, $g(x) = \bar{g}(x)$
la noterons $\bar{g} \bigcup (e, e')$. Ainsi

$$\mathfrak{B}(E, F) = \bigcup_{f \in F} \mathfrak{B}_f \tag{1}$$

où $\mathfrak{B}_f$ désigne l'ensemble des $\bar{g} \bigcup (e, f)$, $\bar{g}$ parcourant l'ensemble $\mathfrak{B}(E-\{e\},F-\{f\})$.
Or les ensembles $\mathfrak{B}_f$ sont disjoints deux à deux et ont le même cardinal, à savoir

$$\text{card } \mathfrak{B}(E - \{e\},F - \{f\}).$$

et

$$B(n) = n.B(n - 1).$$

Récapitulons ces formules :

$\mathfrak{B}(E,F)$=
si $n = 0$ **alors**
$\{\phi\}$
sinon
$\bigcup_{f \in F} \mathfrak{B}_f$

et

$B(n)$=
si $n = 0$ **alors**
1
sinon
$n * B(n - 1)$

3.1 Programme Python

```
# L'ensemble des bijections  de E dans F, sera représenté par une liste d'ensembles.
# Le principe est le suivant:
# On choisit aléatoirement un élément e de E, pour toutes les images f de F possibles, on a
```

```
# couple (e,f) à toutes les bijections de E-{e} dans F-{f}.


def choisir_elmt(E):  # choix aléatoire d'un élément de l'ensemble e
    lst=list(E)
    i=int(len(E)*random())
    return lst[i]

def ajout (elmt,lst_ens): # ajoute elmt à chacun des ensembles de la liste
    if len (lst_ens)==0: # ici ce ne sera jamais le cas!
        return [set(elmt)]
    else:
        res=[]
        for x in lst_ens:
            res= res + [set([elmt])|x]# Attention! il faut écrire [elmt]
        return res


def bij(E,F):  #construit la liste des bijections de E dans F
    if len(F)==0:
        return [set([])]
    else:
        e=choisir_elmt(E)
        res=[]
        for f in F:
            res=res +  ajout ((e,f),bij(E-set(e),F-set(f)))
        return res

def nb_bij(n):  #dénombre les bide E dans F
      if n==0:
          return 1
      else:
          return n*nb_bij(n-1)

 #      **** Procédures pour l'affichage ****

def affiche_ens (E):
    print "{",
    lst=list(E)
    if len(E)>1:
       for i in xrange(0,len(lst)-1):
           print lst[i],",",
    print lst[len(lst)-1],
    print "}"

def affiche_list_ens (L):
    print "[",
    for i in xrange(0,len(L)-1):
        affiche_ens (L[i])
        print",",
    affiche_ens(L[len(L)-1])
    print"]"


#Programme Principal
from random import*
E=set(raw_input("Donner E, un ensemble de lettres : "))
F=set(raw_input("Donner E', un ensemble de chiffres : "))
print "Voici la liste des bijections de E dans F: "
```

```
affiche_list_ens(bij(E,F))
print "Nombre de bijections : ", nb_bij(len(E))
```

Voici le résultat d'une exécution de ce programme dans le cas où l'on veut énumérer et compter le nombre de bijections d'un ensemble E de trois lettres a,b,c dans un ensemble F de trois chiffres 1,2,3

```
Donner E, un ensemble de lettres : abc
Donner F, un ensemble de chiffres : 123
[ { ('b', '1') , ('a', '2') , ('c', '3') }
, { ('b', '1') , ('c', '2') , ('a', '3') }
, { ('c', '1') , ('b', '3') , ('a', '2') }
, { ('b', '3') , ('a', '1') , ('c', '2') }
, { ('a', '1') , ('b', '2') , ('c', '3') }
, { ('c', '1') , ('b', '2') , ('a', '3') }
]
Nombre de bijections :  6
```

4 Enumeration et dénombrement des surjections

Soient E et F deux ensembles finis de cardinaux respectifs n et p, notons $\mathscr{S}(E,F)$ l'ensemble des applications surjectives de E sur F et S_n^p son cardinal. Il est clair que :

$$\mathscr{S}(E,F)=\phi \text{ si } p > n$$

et donc que :

$$S_n^p=0 \text{ si } p > n.$$

De plus si $p = 0$ et $n \neq 0$ nous avons également

$$\mathscr{S}(E,F)=\phi,$$

car il n'y a pas d'application de E sur F, et donc :

$$S_n^0=0$$

Enfin si $p = 0$ et $n = 0$:

$$\mathscr{S}(E,F)=\{\phi\},$$

et donc :

$$S_0^0=1$$

Ces cas singuliers étudiés, envisageons maintenant le cas où $1 \leq p \leq n$, et comme précédemment, soit e un élément de E que nous particularisons. L'ensemble de surjections de E sur F se partitionne en deux sous-ensembles de la façon suivante : le premier est constitué par des surjections pour lesquelles l'image de e n'est l'image d'aucun autre élément de E, et le second est constitué par les surjections pour lesquelles l'image de e est l'image d'au moins un autre élément de E.

Ainsi, pour énumérer $\mathscr{S}(E,F)$, nous énumérerons successivement ces deux sous-ensemble que nous noterons $\mathscr{S}_1$ et $\mathscr{S}_2$.

Si $g \in \mathscr{S}_1$, si nous appelons f l'image de e par g (i.e, $f = g(e)$) et si nous enlevons le couple $((e, f)$ du graphe de g, nous obtenons le graphe d'une surjection de $E - \{e\}$ sur $F - \{f\}$. Cette correspondance est biunivoque. En effet si inversement, $f \in F$ et une surjection $\bar{g}$ de $E - \{e\}$ sur $F - \{f\}$ sont données, il existe une et une seule surjection g de E sur F qui vérifie :
$g(e) = f$ et ($\forall x \in E - \{e\}, g(x) = \bar{g}(x)$).
Nous la noterons $\bar{g} \bigcup \{(e, f)\}$. Par conséquent, nous avons :

$$\mathscr{S}_1 = \bigcup_{f \in F} \bar{g} \bigcup \{(e, f)\} \tag{2}$$

où $\bar{g}$ décrit $\mathscr{S}(E - \{e\},F - \{f\})$
Les ensembles figurant au deuxième membre sont disjoints deux à deux, et ont tous même cardinalité S_{n-1}^{p-1}. Nous obtenons donc

$$\text{card } \mathscr{S}_1 = pS_{n-1}^{p-1}.$$

De la même façon, si $g \in \mathscr{S}_2$, si nous appellons f l'image de e par g, et si nous enlevons le couple (e, f) du graphe de g, nous obtenons le graphe d'une surjection de $E - \{e\}$ sur F, et la correspondance est biunivoque. En effet, si inversement $f \in F$ et une surjection $\bar{g}$ de $E - \{e\}$ sur F sont données, il existe une et une seule surjection g de E sur F qui vérifie :

$$g(e) = f \text{ et } (\forall x \in E - \{e\}, g(x) = \bar{g}(x)).$$

Nous avons donc :

$$\mathscr{S}_2 = \bigcup_{f \in F} \bar{g} \bigcup \{(e, f)\} \tag{3}$$

où $\bar{g}$ décrit $\mathscr{S}(E-\{e\},F)$
Les ensembles qui figurent au second membre sont disjoints deux à deux et sont tous de même cardinalité. Cette cardinalité commune est cette fois-ci S_{n-1}^{p}.
Par conséquent nous avons :
card $\mathscr{S}_2$=pS_{n-1}^{p}.
Si maintenant nous récapitulons les résultats que nous avons obtenus pour les énumérations et pour les dénombrements, nous obtenons :

$\mathscr{S}(E,F)$=
si $p=0$ **alors**
si $n=0$ **alors**
$\{\phi\}$
sinon
ϕ
sinon
si $p>n$ **alors**
ϕ
sinon
$\mathscr{S}_1 \bigcup \mathscr{S}_2$

et

S_n^p=
si $p=0$ **alors**
si $n=0$ **alors**
1
sinon
0
sinon
si $p>n$ **alors**
0
sinon
pS_{n-1}^{p-1} + pS_{n-1}^{p}

Comme on peut le constater l'énumération et le dénombrement des surjections est semblable à celui utilisé pour les combinaisons. On obtient ici des formules récursives à peine plus complexes que celles établies pour les combinaisons. En effet dans le cas des surjections nous avons examiné le cas où $n=0$, ce qui complique légèrement l'écriture des formules d'énumération et de dénombrement.
On peut établir une formule, non récursive, qui permet de calculer le nombre des surjections, mais les outils et le mode de raisonnement mis en jeu pour établir cette formule est beaucoup plus complexe que le raisonnement récursif développé ci-dessus. Pour établir cette formule on utilise la formule de Poincaré pemettant

de calculer la cardinalité d'une union d'ensembles non disjoints. L'idée principale est de dire que le nombre d'applications surjectives est égal au nombre d'applications diminué du nombre d'applications qui ne sont pas surjectives. la formule obtenue est la suivante :

$$S_n^p = p^n - \sum_{k=1}^{p} (-1)^{k-1} C_p^k (p-k)^n \tag{4}$$

4.1 Programme Python

```
  # Enumération et dénombrement des surjections de E sur F

# On représente une surjection par un ensemble de couples
# L'ensemble des surjections sera représenté par une liste d'ensembles.
# Le principe est le suivant:
# On choisit un élément e de E, l'ensemble des surjections est partitionné en 2:
# les surjections où l'image f de e n'est atteinte que par e et celle où
# l'image f de e est atteinte par au moins un autre point.
# Ainsi pour toutes les images f de e on ajoute le couple (e,f) à toutes
# les surjections de E-{e} sur F-{f} et à toutes les surjections de E-{e} sur F.

def choisir_elmt(E):  # choix aléatoire d'un élément de l'ensemble e
    lst=list(E)
    i=int(len(E)*random())
    return lst[i]

def ajout (elmt,lst_ens): # ajoute elmt à chacun des ensembles de la liste
    if len (lst_ens)==0:
        return [set([elmt])]
    else:
        res=[]
        for x in lst_ens:
            res= res + [set([elmt])|x]    # Attention! il faut écrire [elmt] et pas elmt
        return res

                                  # énumération
def surj(E,F):
    if len(F)==0:
        if len(E)==0:
            return [set([])]
        else:
            return []
    else:
        if len(F)>len(E):
            return []
        else:
            e=choisir_elmt(E)
            res=[]
            for f in F:
                res=res + ajout((e,f),surj(E-set(e),F-set(f))+surj(E-set(e),F))
            return res

                                  # dénombrement
```

```
def nb_surj(n,p):
    if p==0:
        if n==0:
            return 1
        else:
            return 0
    else:
        if p>n:
            return 0
        else:
            return p*(nb_surj(n-1,p-1)+nb_surj(n-1,p))

 #      **** Procédures pour l'affichage ****

def affiche_ens (E): # affiche un ensemble comme en math
    print "{",
    lst=list(E)
    if len(E)>1:
       for i in xrange(0,len(lst)-1):
           print lst[i],",",
    print lst[len(lst)-1],
    print "}"

def affiche_list_ens (L):  # affiche une liste d'ensemble
    print "[",
    for i in xrange(0,len(L)-1):
        affiche_ens (L[i])
        print",",
    affiche_ens(L[len(L)-1])
    print"]"

#Programme Principal
from random import*

E=set(raw_input("Donner E, un ensemble de lettres : "))
F=set(raw_input("Donner F, un ensemble de chiffres : "))
print" Voici la liste des surjections : "
affiche_list_ens surj(E,F)

print "Nombre de surjections : ", nb_surj(len(E),len(F))
```

Voici le résultat d'une exécution de ce programme dans le cas où l'on veut énumérer et compter le nombre de surjections d'un ensemble E de trois lettres a,b,c sur un ensemble F de deux chiffres 1,2

```
Donner E, un ensemble de lettres : abc
Donner F, un ensemble de chiffres : 12
Voici la liste des surjections
[ { ('a', '1') , ('c', '2') , ('b', '2') }
, { ('c', '1') , ('a', '1') , ('b', '2') }
, { ('b', '1') , ('a', '1') , ('c', '2') }
```

```
, { ('b', '1') , ('c', '1') , ('a', '2') }
, { ('c', '1') , ('a', '2') , ('b', '2') }
, { ('b', '1') , ('a', '2') , ('c', '2') }
]
Nombre de surjections :  6
```