# Le triangle de Pascal

## Présentation

### Dans le programme (spécialité Terminale)

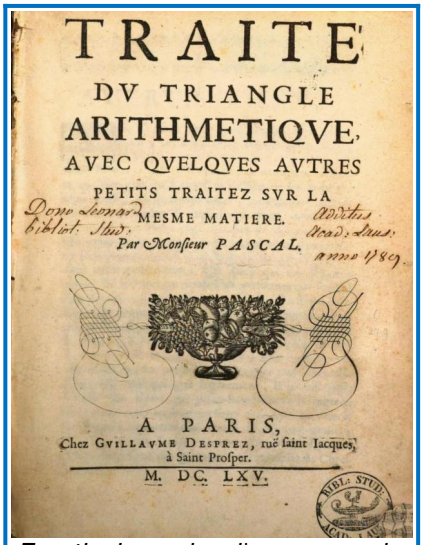
| **Contenus** | **Capacités attendues** | **Algorithmes** |
|---|---|---|
| Pour $0 \leq k \leq n$, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.<br><br>Relation et triangle de Pascal. | Dans le cadre d'un problème de dénombrement, utiliser une représentation adaptée et reconnaître les objets à dénombrer.<br>Effectuer des dénombrements simples.<br>Démonstration par dénombrement de la formule $\sum_{k=0}^{n} \binom{n}{k} = 2^n$. | Génération de la liste des coefficients $\binom{n}{k}$ à l'aide de la relation de Pascal. |

### Situation déclenchante

En mathématiques, les coefficients binomiaux, définis pour tout entier naturel $n$ et tout entier naturel $k$ inférieur ou égal à $n$, donnent le nombre de sous-ensembles différents à $k$ éléments que l'on peut former à partir d'un ensemble contenant $n$ éléments. Les coefficients binomiaux sont utilisés dans de nombreux domaines : binôme de Newton, dénombrement, développement en série, probabilités, etc.…

En 1654, Blaise Pascal écrit son Traité du triangle arithmétique dans lequel il donne une présentation pratique en tableau des coefficients du binôme, le « triangle arithmétique », maintenant connu sous le nom de « triangle de Pascal ».

Il faut noter qu'un mathématicien chinois sous la dynastie des Qin, Yang Hui, avait travaillé quatre siècles plus tôt sur un concept semblable au triangle de Pascal.


TRAITE
DV TRIANGLE
ARITHMETIQVE,
AVEC QVELQVES AVTRES
PETITS TRAITEZ SVR LA
MESME MATIERE.
Par Monsieur PASCAL.
A PARIS,
Chez GVILLAVME DESPREZ, ruë saint Iacques,
à Saint Prosper.
M. DC. LXV.

*Frontispice de l'ouvrage de Blaise Pascal (1665). Crédit : archive.org*

### Buts à atteindre

1 Écrire une fonction qui prend en paramètres deux entiers naturels $n$ et $k$ et renvoie le coefficient binomial $k$ parmi $n$. Cette question est une opportunité pour effectuer une activité de groupe.

**Travail de groupe :** chaque groupe de 2 ou 3 élèves choisit une définition ou propriété caractéristique des coefficients binomiaux et en tire un algorithme de calcul de ceux-ci, puis le programme en langage Python. Ensuite, les groupes échangent leurs productions et en vérifient l'exactitude et l'efficacité. Les élèves pourront choisir parmi les définitions et propriétés suivantes :

$\binom{n}{k} = \frac{k!}{k!(n-k)!}$ ; $\binom{n}{0} = \binom{n}{n} = 1$ et $\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!}$ pour $n \in \mathbb{N}^*, k \in \mathbb{N}^*$ ; $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ ; $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ .

*L. DIDIER & R. CABANE*

2 Nombre de parties à $k$ éléments d'un ensemble de $n$ éléments : écrire une fonction prenant en paramètre un entier naturel $n$, permettant de construire le triangle de Pascal jusqu'à la ligne de rang $n$, et renvoyant la liste des listes des coefficients binomiaux.

3 Écrire une fonction qui prenant comme paramètre un entier naturel $n$ et renvoyant la liste des $\sum_{k=0}^{i} \binom{i}{k}$ ($i$ variant de 0 à $n$). Quelle conjecture peut-on faire ?

*L. DIDIER & R. CABANE*

# Fiche méthode

## Proposition de résolution

**Préalable informatique :**

L'opérateur de division `/` en Python renvoie toujours un nombre à virgule (ou « flottante »). Comme on s'attend bien à ce que les coefficients binomiaux soient des entiers, on emploiera ici la « division entière » `//`, qui renvoie le quotient de la division euclidienne (`a//b` est la partie entière de `a/b`). De fait, nous ne l'emploierons que pour des divisions « tombant juste ».

**Pour atteindre l'objectif 1 :**

Dans cette partie nous allons présenter différentes manières pour calculer les coefficients binomiaux en essayant de les comparer. La fonction `binome` prend comme paramètres deux entiers naturels $n$ et $k$ et renvoie le coefficient binomial associé.

**Pour atteindre l'objectif 2 :**

Une fonction `trianglepascal` qui prend comme argument un entier naturel $n$ et qui renvoie la liste des listes des coefficients binomiaux en affichant le triangle de Pascal.

**Pour atteindre l'objectif 3 :**

Une fonction `conjecture` prenant comme argument un entier naturel $n$ et renvoyant la liste des $\sum_{k=0}^{i}\binom{i}{k}$ pour tout $i$ entier naturel $(0 \leq i \leq n)$.

PYTHON SHELL

```
>>> trianglepascal(5)
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, [1, 1], [1, 2, 1], [1, 3, 3,
 1], [1, 4, 6, 4, 1], [1, 5, 10,
 10, 5, 1]]
```

PYTHON SHELL

```
>>> # L'exécution de PASCAL
>>> from PASCAL import *
>>> conjecture(5)
[1] somme= 1
[1, 1] somme= 2
[1, 2, 1] somme= 4
[1, 3, 3, 1] somme= 8
[1, 4, 6, 4, 1] somme= 16
[1, 5, 10, 10, 5, 1] somme= 32
[1, 2, 4, 8, 16, 32]
```

## Objectif 1 : étapes de résolution

Voici cinq propositions correspondant aux différentes définitions.

ÉDITEUR : BINOM
LIGNE DU SCRIPT 0010

```
def facto(n):
    f=1
    for i in range(1,n+1):
        f=f*i
    return f

def binom1(n,k):
    X=(facto(n)//facto(k))//facto(
        n-k)
    return X
```

La fonction `binom1` est fondée sur la formule de définition des coefficients. Elle pose le problème de la taille des factorielles dès que `n` dépasse 70. Noter l'usage de parenthèses pour fixer l'ordre des opérations arithmétiques.

ÉDITEUR : BINOM
LIGNE DU SCRIPT 0020

```
def binom2r(n,k):
    if k==0:
        return 1
    return binom2r(n,k-1)*(n-k+1)/
        /k

def binom2i(n,k):
    X=1
    for i in range(1,k+1):
        X=X*(n-i+1)//i
    return X
```

La fonction `binom2r` est basée sur un appel récursif[1], au contraire de la fonction `binome2i` (itérative). Tout se fonde sur la formule usuelle de définition des coefficients.

ÉDITEUR : BINOM
LIGNE DU SCRIPT 0032

```
def binom3r(n,k):
    if k==0:
        return 1
    return binom3r(n-1,k-1)*n//k

def binom3i(n,k):
    X=1
    for j in range(1,k+1):
        X=(X*(n-k+j))//j
    return X
```

La fonction `binom3r` est basée sur un appel récursif inspiré de la formule $\binom{n}{k}=\frac{n}{k}\binom{n-1}{k-1}$.

Une version itérative est proposée dans `binom3i`.

1 C'est-à-dire, une fonction qui s'appelle elle-même.

La fonction `binome4` (ci-contre) permet le calcul de coefficients binomiaux par la formule de Pascal avec des appels récursifs.

La fonction `binome5` permet le calcul des coefficients binomiaux par la formule de Pascal et sans appels récursifs ; elle s'appuie sur un calcul du triangle de Pascal, ligne par ligne. Pour éviter d'avoir à garder en mémoire deux lignes du triangle, on réécrit la ligne en cours avec les coefficients nouvellement calculés, ce qui fonctionne si on procède *de droite à gauche* et en initialisant la liste `L` avec `n` valeurs toutes nulles.

`[0]*n` fournit liste de zéros de longueur n.

`range(1,0,-1)` produit ici des valeurs qui « descendent » (voir annexe 1).

*Exécutons les différentes fonctions.*

Lors des différentes exécutions laquelle semble la plus rapide ?

*Indication :* la récursivité peut consommer beaucoup de temps de calcul en raison des appels de fonction qui se multiplient énormément.

### ▶ Pour atteindre l'objectif 2 :

Le programme utilise les listes que l'on construit ligne par ligne en utilisant la formule du triangle de Pascal. La liste `p` va contenir les coefficients binomiaux de *i* parmi *k* pour *i* allant de 0 à *k*.

La liste `p2` sera utilisée pour construire la ligne du dessous (en tant que liste de stockage) dans le triangle de Pascal. La liste `r` contiendra les listes des coefficients binomiaux.

### ▶ Pour atteindre l'objectif 3 :

Pour calculer les différentes sommes, il suffit d'effectuer la somme des coefficients de chaque ligne du triangle de Pascal.

On peut donc reprendre le programme précédent et rajouter une liste `somme` qui va être construite en effectuant à chaque passage de boucle la somme des coefficients de la liste `p`.

Une commodité : pour une liste de nombres `L`, l'expression `sum(L)` donne la somme des termes de `L` (cf. appendice 1).

ÉDITEUR : BINOM
LIGNE DU SCRIPT 0042

```
def binom4(n,k):
  if n==0 or k==0 or n==k:
    return 1
  return binom4(n-1,k-1)+binom4(
      n-1,k)
```

ÉDITEUR : PASCAL2
LIGNE DU SCRIPT 0051

```
def LignePascal(n):
  n=n+1
  L=[0]*n
  L[0]=1
  for i in range(n):
    for j in range(i,0,-1):
      L[j]=L[j-1]+L[j]
  return L

def binome5(n,k):
  return LignePascal(n)[k]
```

PYTHON SHELL

```
>>> binom1(26,17)
3124550
>>> binom1(23,17)
100947
>>> binom2r(23,17),binom2i(23,17
)
(100947, 100947)
>>> binom3r(23,17),binom3i(23,17
)
(100947, 100947)
```

ÉDITEUR : PASCAL
LIGNE DU SCRIPT 0002

```
def trianglepascal(n):
  p=[1]
  r=[1]
  print(p)
  k=1
  while k<=n:
    k=k+1
    p2=[1]
    for i in range(1,k-1):
      p2.append(p[i-1]+p[i])
    p=p2
    p.append(1)
    r.append(p)
    print(p)
  return r
```

Initialisations

Relation de Pascal

ÉDITEUR : PASCAL
LIGNE DU SCRIPT 0054

```
def conjecture(n):
  p=[1]
  somme=[1]
  print(p,"somme=",1)
  k=1
  while k<=n:
    k=k+1
    p2=[1]
    for i in range(1,k-1):
      p2.append(p[i-1]+p[i])
    p=p2
    p.append(1)
    print(p,"somme=",sum(p))
    somme.append(sum(p))
  return(somme)
```

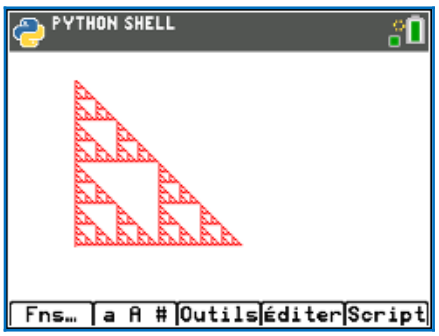
Fns… | a A # | Outils | Exéc | Script

# Pour aller plus loin

## Prolongement possible (spécifique TI-83)

Dans cette partie nous allons faire apparaître les triangles de Sierpinski en utilisant la parité des coefficients binomiaux.

> Écrire un script Python permettant d'afficher le triangle de Pascal dans lequel on remplacera les coefficients par un pixel rouge si le coefficient binomial est impair et un pixel blanc (ou rien) si le coefficient binomial est pair.




## Code Python proposé

**TI-83** Le module[2] `ti_draw` permet d'utiliser les instructions associées à l'affichage des pixels.

Les commandes graphiques employées sont simples, voir l'annexe 2 pour plus de détails.

```
ÉDITEUR : SIERPNS2
LIGNE DU SCRIPT 0010
from ti_draw import *
def trpsc(n):
    clear()
    set_color(255,0,0)
    p=[1] # ligne 0 du triangle
    for k in range(1,n):# ligne k
        p2=[1] # colonne 0
        for i in range(1,k): #col i
            p2.append(p[i-1]+p[i])
        p=p2+[1] # colonne k+1
        for i in range(k+1):
            if p[i]%2==1:
                plot_xy(50+i,50+k,7)
    show_draw()

trpsc(64)
```

On efface dans un premier temps l'écran (clear).

On fixe la couleur du tracé au rouge (set_color).

On calcule les coefficients binomiaux comme dans l'objectif 2.

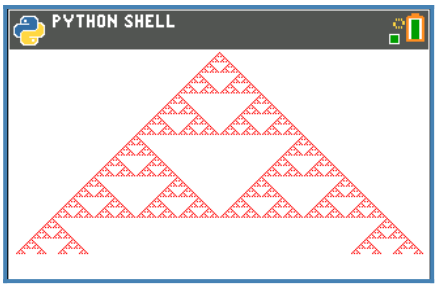
Une fois les coefficients d'une ligne calculés, en fonction de la parité, on allume (plot_xy) certains pixels en rouge.

On met en attente (show_draw).

2 Ce module (ou bibliothèque) est accessible comme module complémentaire avec le système 5.7.

*L. DIDIER & R. CABANE*

## Un défi




On peut souhaiter mieux « orienter » le triangle de Sierpinski et obtenir le schéma ci-contre. Comment faire ?

Une première idée est de créer une fonction d'accès aux « pixels » requis pour chaque couple $(i,j)$ dont on veut représenter le coefficient binomial (fonction `pt`, ci-contre). Cela ne change pas l'algorithme et permet de mieux séparer le calcul de l'affichage.

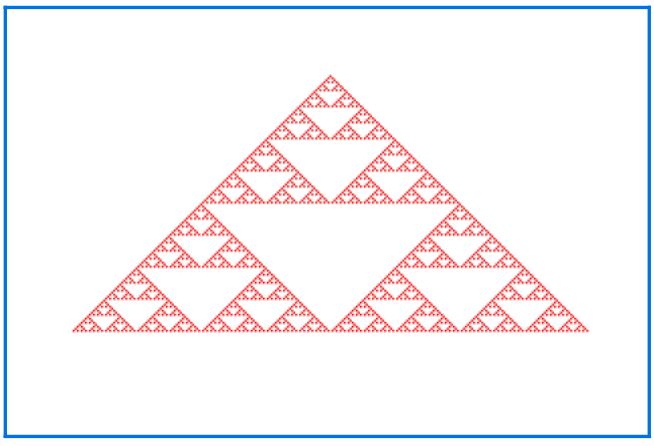
Tant qu'à faire, le calcul est simplifié en recourant à l'algorithme de la fonction `binome5`.

Remarque : il n'est pas nécessaire d'afficher de point quand le coefficient est pair ; on teste donc l'imparité du nombre `L[j]` (qui contient la valeur de $\binom{i}{j}$) en testant si le reste de sa division par 2 est égal à 1, ce qui s'écrit `L[j]%2==1`.

ÉDITEUR : SIERPNS3
LIGNE DU SCRIPT 0011

```
from ti_draw import *
def pt(i,j):
  plot_xy(160-i+2*j,32+i,7)
def t(p):
  clear()
  set_color(255,0,0)
  L=[1]+[0]*p # ligne 0
  for i in range(p+1):# ligne i
    for j in range(i,-1,-1):
      if j>0:
        L[j]=L[j-1]+L[j]
      if L[j]%2==1:
        pt(i,j)
  show_draw()
t(64)
```

Voici quelques copies d'écran réalisées avec le logiciel Nspire™ CX qui reprennent les algorithmes précédents déclinés sur la calculatrice TI Nspire™ CX II-T.



```
from ti_draw import *
def pt(i,j):
  plot_xy(160-i+2*j,32+i,7)
def t(p):
  clear()
  set_color(255,0,0)
  n=p+1
  L=[0]*n
  L[0]=1
  for i in range(n):
    for j in range(i,-1,-1):
      if j>0:
        L[j]=L[j-1]+L[j]
      if L[j]%2==1:
        pt(i,j)
```