Math93.com

TD 1 - Terminale Spécialité Maths
Algorithmique
Combinatoire, Dénombrement

Les algorithmes dont l'intitulé est précédé de ROC sont explicitement indiqués dans le programmes de terminale spécialité mathématiques. Ils sont à connaître.

Exercice 1. Fonction factorielle

On note $n!$ (se lit « factoriel n ») le nombre $1 \times 2 \times 3 \times \cdots \times n$, pour tout entier naturel $n > 0$. Par convention on définit :

$$\begin{cases} 0! = 1 \\ n! = 1 \times 2 \times 3 \times \cdots \times n \quad , n \in \mathbb{N}^* \end{cases}$$

Remarque historique

La **notation factorielle** est introduite par le mathématicien Christian KRAMP (1760-1826) en 1808 dans *Éléments d'arithmétique universelle* (1808).

1. Calculer $1!, 2!, 3!, 4!$ et $5!$.
2. Écrire une fonction **fact(n)** qui renvoie $n!$, avec $n \geqslant 0$.

```
# Dans l'éditeur PYTHON
def fact(n):
    '''In : indice n, entier naturel (int)
       Out : n!'''
    assert n>=0
    ...
    return ..
```

3. Avec le module math.

Aide

Factorielle : $n! = 1 \times 2 \times 3 \times \ldots \times n$.
Ce produit ce nomme factoriel n et se note $n!$.
Avec le *module math*, il existe un fonction en python qui le calcul directement : *math.factorial(n)*.
Pour l'appeler, il faut charger le *module math* au début du programme (ligne 1),
la syntaxe est *import math*.

Chargez le module math en ligne 1 en écrivant *import math* puis vérifiez que *math.factorial(n)* donne bien le même résultat que votre fonction pour quelques valeurs de n.

Exercice 2. ROC : Triangle de Pascal 1

Propriété 1 (Relation de Pascal)

Soit n et p des entiers naturels tels que $1 \leqslant p \leqslant n-1$.

$$\boxed{\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}}$$

1. Compléter rapidement ce tableau en utilisant la relation de Pascal.

n/p	$p=0$	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$	$p=7$
$n=0$	1							
$n=1$	1	1						
$n=2$	1		1					
$n=3$	1			1				
$n=4$	1				1			
$n=5$	1					1		
$n=6$	1						1	
$n=7$	1							1

2. On cherche à écrire une fonction python $Pascal(n)$ de paramètres n et qui renvoie la liste des coefficients $\binom{n}{p}$ pour n entier fixé en utilisant la relation de Pascal.

- (L5) Pour cela on va initialiser une liste L correspondant à la ligne $n=0$ du triangle de Pascal.
- (L6) Puis on copie cette liste dans une autre liste M. Attention, pour copier deux listes, la syntaxe est particulière (sinon elles seront modifiées simultanément).
- (L8) On fait une première boucle sur les lignes de $ligne_n = 1$ à n (attention aux indices ...)
- (L9) on ajoute 1 à le liste M.
- (L10) on boucle sur les colonnes de $p=1$ à $n-1$ car on sait que la première et la dernière valeur du tableau sont des 1 (attention aux indices ...)

```
1  def Pascal(n):
2      '''IN : n entier
3        OUT : liste des coefficients binomiaux Cnp, pour n fixé'''
4      assert n==int(n) # pour s'assurer que n est entier
5      L=[1]  # initialisation de la liste
6      M=L[:] # ou M=list(L) pour copier des listes il faut utiliser cette
7             # syntaxe sinon les 2 seront modifiées simultanément
8      for ligne_n in range (... , ...):
9          M.append(1) # on ajoute 1 à la fin
10         for colonne_p in range (... , ...):
11             k=colonne_p
12             M[k]= ..... # Formule de Pascal  Cn,k=Cn-1,k-1 + Cn-1,k
13         L=M[:] # pour copier des listes ... Attention
14     return L
```

3. Tester votre fonction avec les valeurs du tableau.
4. Écrire une fonction $triangle(n)$ de paramètre un entier n , utilisant la fonction précédente $Pascal(n)$, qui renvoie le triangle de Pascal complet sous forme de listes.
5. Écrire une fonction $cnp(n,p)$ de paramètre un entier (n,p) , utilisant la fonction précédente $Pascal(n)$, qui renvoie le le coefficient binomiale $\binom{n}{p}$ en testant des conditions de validité de p.

Exercice 3. ROC : Triangle de Pascal 2

On propose ici une autre fonction $Pascal2(n)$ de paramètres n et qui renvoie la liste des coefficients $\binom{n}{p}$ pour n entier fixé en utilisant la relation de Pascal.

```
1  def Pascal2(n):
2      '''IN : n entier
3        OUT : liste des coefficients binomiaux Cnp, pour n fixé'''
4      assert n==int(n)
5      L=[1]
6      for i in range (1,n+1):
7          L.append(1)
8          a=L[0]
9          b=L[1]
10         for k in range(1,i):
11             L[k]=a+b
12             a=b
13             b=L[k+1]
14     return L
```

1. Tester cette fonction pour quelques valeurs de n.
2. On va chercher à comprendre le programme. Pour $n=4$ on va écrire chacune des variables.

   - Pour $i=1$, après la ligne 7 :
     - (L7) $L=[1,1]$;
     - (L8) $a=1=L[0]$;
     - (L9) $b=1=L[1]$.

     On n'entre pas dans la seconde boucle.
   - Pour $i=2$, après la ligne 7 :
     - (L7) $L=[1,1,1]$;
     - (L8) $a=1=L[0]$;
     - (L9) $b=1=L[1]$.

     La variable k de la seconde boucle ne prend que la valeur 1.

k (L10)	$L[k]$ (L11)	Formule	a (L12)	b (L13)
1	$L[1]=1+1=2$	$\binom{1}{0}+\binom{1}{1}=\binom{2}{1}$	1	$b=L[2]=1$

- Pour $i = 3$, après la ligne 7 :
  - (L7) $L = [1, 2, 1]$;
  - (L8) $a = 1 = L[0]$;
  - (L9) $b = 2 = L[1]$.

  La variable k de la seconde boucle prend les valeur 1 à 2 .

k (L10)	$L[k]$ (L11)	Formule	a (L12)	b (L13)
1	$L[1] = \cdots$	$\binom{\cdots}{\cdots} + \binom{\cdots}{\cdots} = \binom{\cdots}{\cdots}$	1	$b = L[2] = \cdots$
2	$L[2] = \cdots$	$\binom{\cdots}{\cdots} + \binom{\cdots}{\cdots} = \binom{\cdots}{\cdots}$	1	$b = L[3] = \cdots$

- Pour $i = 4$, après la ligne 7 :
  - (L7) $L = \cdots$;
  - (L8) $a = \cdots$;
  - (L9) $b = \cdots$.

  La variable k de la seconde boucle prend les valeur 1 à 3 .

k (L10)	$L[k]$ (L11)	Formule	a (L12)	b (L13)
1	$L[1] = \cdots$	$\binom{\cdots}{\cdots} + \binom{\cdots}{\cdots} = \binom{\cdots}{\cdots}$	1	$b = L[2] = \cdots$
2	$L[2] = \cdots$	$\binom{\cdots}{\cdots} + \binom{\cdots}{\cdots} = \binom{\cdots}{\cdots}$	1	$b = L[3] = \cdots$
3	$L[3] = \cdots$	$\binom{\cdots}{\cdots} + \binom{\cdots}{\cdots} = \binom{\cdots}{\cdots}$	1	$b = L[4] = \cdots$

Exercice 4. ROC : Triangle de Pascal 3

On propose ici une autre fonction $triangle2(n)$ de paramètres n et qui renvoie la liste des lignes du triangle de Pascal de 0 à n.
La ligne 2 permet de construire un tableau de $(n + 1)$ lignes et $(n + 1)$ colonnes.

```
1  def Triangle(n):
2      t=[[0 for i in range(n+1)] for i in range(n+1)]
3      for i in range(n+1):
4          t[i][0]=1
5          for j in range(1,i+1):
6              t[i][j]=...
7              t[i][i]=1
8      return t
9
10 from pprint import pprint # pour avoir un affichage en ligne
11 pprint(Triangle(10))
```

1. Compléter la ligne 6 qui correspond à l'application de la formule de Pascal.
2. Testez cette fonction pour quelques valeurs de n.
3. Expliquez les lignes 4 et 7.

Exercice 5. ROC : Tirage aléatoire d'une permutation

TP1 page 40 du manuel Indice de Bordas.

Exercice 6. ROC : génération des parties de 2 et 3 éléments d'un ensemble

TP2 page 40 du manuel Indice de Bordas.

Exercice 7. Factorielle : une approche de la récursivité (Hors programme)

Un algorithme est dit récursif si il s'appelle lui-même.
La factorielle se définit pour des entiers naturels de la fonction suivante :

$$\begin{cases} 0! = 1 \\ n! = 1 \times 2 \times 3 \times \cdots \times n \quad , n \in \mathbb{N}^* \end{cases} \qquad \text{ou} \qquad n! = n \times (n-1)!$$

Donc pour définir la fonction qui calcule la factorielle de n, il suffit d'appeler cette même fonction mais en lui demandant de calculer la factorielle de $(n-1)$, et multiplier le résultat par n.
La factorielle de $(n-1)$ sera calculée en calculant la factorielle de $(n-2)$ et ainsi de suite. Il suffit juste de définir que la factorielle de 1 est 1 pour "arrêter la récursion" et ne plus appeler la fonction qui calcule la factorielle.
Vérifier alors que ce programme calcule bien la factorielle de n.

```
def g(n):
    '''In : indice n, entier naturel
      Out : n!'''
    assert n>=0
    if n==0:
            return 1
    else:
            return n*g(n-1)
```

↩ **Fin du TD** ↪