# Travaux Pratiques n°3
# Corrigé

▷ **Exercice 1.**

```
A=list(range(10,21))
B=list(range(-15,16,2))
C=list(range(0,200,7))
D=list(range(14,0,-1))
E=25*[3]
F=[]
for n in range(21):
    for k in range(n):
        F.append(n)

print(A,B,C,D,E,F,sep='\n')
print(len(F))              # 210
print(F[150])              #  17
```

▷ **Exercice 2.**

```
def Occurrences(a,L):
    """Compte le nombre d'occurrences de a dans L."""
    n=0
    for k in L:
        if a==k:
            n=n+1
    return n

print(Occurrences(1,[4,6,5,7,1,3,2,1]))
print(Occurrences(8,[4,6,5,7,1,3,2,1]))
print(Occurrences(8,[8,8]))
print(Occurrences(8,[]))
```

▷ **Exercice 3 : Coefficients du binôme.**

On suppose que la fonction `factorielle` est déjà définie.

a.

```
def CoefBino(n,k):
    """Calcule (k parmi n)."""
    if 0<=k and k<=n:
        return factorielle(n)//factorielle(k)//factorielle(n-k)
    return 0
```

Les tests :

```
for [n,k] in [[0,0],[3,1],[6,9],[50,13],[10000,9997]]:
    print(n,k,CoefBino(n,k))
for k in range(5):
    print(CoefBino(4,k))
```

On sait que les coefficients du binôme sont tous entiers, donc que le quotient $\frac{n!}{k!(n-k)!}$ est entier. Ainsi la division entière donne le même résultat que la division classique. Mais la première renvoie un entier, alors que la seconde renvoie un flottant, ce qui est inutile ici et alourdit l'écriture. Par exemple pour $\binom{3}{1}$ il vaut mieux obtenir `3` que `3.0`.

b.

```
def LigneSuivante(L):
    M=[L[0]]
    for k in range(len(L)-1):
        M.append(L[k]+L[k+1])
    M.append(L[-1])
    return M

print(LigneSuivante([3,5,-1,0,7]))          # [3,8,4,-1,7,7]
```

c.

```
N=int(input("Combien de lignes voulez-vous ? "))

L=[1]
print(L)
for n in range(N):
    L=LigneSuivante(L)
    print(L)
```

## ▷ Exercice 4 : Nombres premiers, algorithme naïf.

a.

```
from math import sqrt

def est_premier(n):
    """Teste si n est premier."""
    if n<2:
        return False
    for k in range(2,int(sqrt(n))+1):
        if n%k==0:
            return False
    return True
```

b.

```
for i in range(21):
    if est_premier(i):
        print(i,end=' ')
print()
```

c.

```
n=0
for i in range(1000):
    if est_premier(i):
        n+=1
print("Nombre de nombres premiers inférieurs à 1000 :",n)
```

On obtient 168 nombres premiers inférieurs à 1000.

d.

```
depart=10**10
k=depart
while not est_premier(k):
    k+=1
print(k,"est le premier nombre premier après",depart)
```

Le plus petit nombre premier au-dessus de $10^{10}$ est $10\,000\,000\,019 = 10^{10} + 19$.

▷ **Exercice 5 : Crible d'Ératosthène.**

a.

```
from math import sqrt

def Crible(N):
    """Construit le crible D'Eratosthène."""
    # Création du crible :
    C=[False]*2+(N-2)*[True]
    # Parcours du crible :
    for n in range(2,int(sqrt(N))+1):
        if C[n]:
            for k in range(n**2,N,n):
                C[k]=False
    return C

C1=Crible(21)
for n in range(21):
    if C1[n]:
        print(n,end=' ')
print()
```

b.

```
20 C1=Crible(1000)
21
22 n=0
23 for b in C1:
24     if b:
25         n+=1
26
27 print("Nombre de nombres premiers inférieurs à 1000 :",n)
```

On obtient de nouveau 168 nombres premiers.

▷ **Exercice 6 : Comparaison des algorithmes.**

a. On exécute le programme suivant sans réinitialisation de la mémoire, de façon à ce que les fonctions `est_premier` et `Crible` soient encore définies.

```
1  from time import clock
2
3  N=100
4
5  t1=clock()
6  C1=[]
7  for k in range(N+1):
8      C1=C1+[est_premier(k)]
9  t2=clock()
10
11 t3=clock()
12 C2=Crible(N)
13 t4=clock()
14
15 print("Temps pour l'algorithme naïf :      ",t2-t1)
16 print("Temps pour le crible d'Eratosthène :",t4-t3)
```

b. On constate que l'algorithme du crible d'Ératosthène est beaucoup plus rapide que le premier. Cette différence s'accentue avec la taille : de l'ordre de 2 000 fois plus rapide si $N = 10^5$.

▷ **Exercice 7 : Nombres parfaits et amis.**

a.

```
def Diviseurs(n):
    """Renvoie la liste des diviseurs de n autres que lui-même."""
    LD=[]
    for k in range(1,n):
        if n%k==0:
            LD.append(k)
    return LD

# for k in [1,2,3,4,12,28,29]:
#     print(Diviseurs(k))
```

b.

```
def Somme(L):
    """Calcule la somme des nombres de la liste L."""
    S=0
    for a in L:
        S+=a
    return S

def Parfait(n):
    """Teste si n est parfait."""
    return Somme(Diviseurs(n))==n

lim=10**4
for n in range(1,lim+1):
    if Parfait(n):
        print(n,"est parfait. ")
```

Les nombres parfaits inférieurs à $10^4$ sont 6, 28, 496 et 8128.

On connaît actuellement 48 nombres parfaits. Il est conjecturé que tout nombre parfait est pair, mais ce n'est pas démontré. On ne sait pas non plus prouver qu'il existe une infinité de nombres parfaits.

c.

```
for n in range(1,lim+1):
    m=Somme(Diviseurs(n))
    if m>n and n==Somme(Diviseurs(m)):
        print(n,'et',m,'sont amis.')
```

Les couples de nombres amis inférieurs à $10^4$ sont $(220, 284)$, $(1184, 1210)$, $(2620, 2924)$, $(5020, 5564)$ et $(6232, 6368)$.

On connaît un grand nombre de couples de nombres amis, mais on ne sait pas prouver qu'il en existe une infinité.

▷ **Exercice 8 : Nombres premiers jumeaux.**

On suppose que la fonction `Crible` est toujours en mémoire.

```
N=2**20
C=Crible(N)
c=0
for k in range(1,N-2,2):    # Inutile de tester les pairs
    if C[k] and C[k+2]:
        print("(",k,",",k+2,")",sep="")
        c+=1
print("Il existe ",c," couples de nombres premiers jumeaux inférieurs
    à ",N,".",sep="")
```

On obtient 8 535 couples de nombres premiers jumeaux inférieurs à $2^{20}$.

▷ **Exercice 9 : Conjecture de Goldbach.**

On suppose de nouveau que la fonction `Crible` est toujours en mémoire.

```
N=2**20
C=Crible(N)

Conjecture=True
affichage=False # contrôle l'affichage des sommes
for n in range(4,N,2): # on parcourt les entiers pairs
    k=2
    while not(C[k] and C[n-k]) and k<=n//2:
        k+=1
    if C[k] and C[n-k]:
        if affichage:
            print(n,"=",k,"+",n-k)
    else:
        print("La conjecture est fausse :",n,"ne la vérifie pas.")
        Conjecture=False
if Conjecture:
    print("La conjecture est vraie au moins jusqu'à",N)
```

La conjecture semble effectivement valide.