



# Python

## Exercices corrigés

## Énoncés des exercices

### Remarque

☞ Les exercices suivants sont fournis à titre d'exemples et de modèles. Ils sont soit simples, soit moins simples (notés ▷ dans la marge) soit difficiles (notés ▷▷).

1. Écrire un programme qui, à partir de la saisie d'un rayon et d'une hauteur, calcule le volume d'un cône droit.
2. Une boucle while : entrez un prix HT (entrez 0 pour terminer) et affichez sa valeur TTC.
3. Une autre boucle while : calculez la somme d'une suite de nombres positifs ou nuls. Comptez combien il y avait de données et combien étaient supérieures à 100. Un nombre inférieur ou égal à 0 indique la fin de la suite.
4. L'utilisateur donne un entier positif  $n$  et le programme affiche PAIR s'il est divisible par 2 et IMPAIR sinon.
5. L'utilisateur donne un entier positif et le programme annonce combien de fois de suite cet entier est divisible par 2.
6. L'utilisateur donne un entier supérieur à 1 et le programme affiche, s'il y en a, tous ses diviseurs propres *sans répétition* ainsi que leur nombre. S'il n'y en a pas, il indique qu'il est premier. Par exemple : <

```
Entrez un entier strictement positif : 12
Diviseurs propres sans répétition de 12 : 2 3 4 6 (soit 4 diviseurs propres)
```

```
Entrez un entier strictement positif : 13 Diviseurs propres sans
répétition de 13 : aucun ! Il est premier
```

7. Écrire un programme qui estime la valeur de la constante mathématique  $e$  en utilisant la formule :

$$e = \sum_{i=0}^n \frac{1}{i!}$$

Pour cela, définissez la fonction factorielle et, dans votre programme principal, saisissez l'ordre  $n$  et affichez l'approximation correspondante de  $e$ .

8. Un gardien de phare va aux toilettes cinq fois par jour. Les WC sont au rez-de-chaussée. Écrire une procédure (donc sans retour) hauteurparcourue qui reçoit deux paramètres le nombre de marches du phare et la hauteur de chaque marche (en cm), et qui affiche :

```
Pour x marches de y cm, il parcourt z.zz m par semaine.
```

## *Exercices*

On n'oubliera pas :

- qu'une semaine comporte 7 jours ;
- qu'une fois en bas, le gardien doit remonter ;
- que le résultat est à exprimer en m.

9. Un permis de chasse à points remplace désormais le permis de chasse traditionnel. Chaque chasseur possède au départ un capital de 100 points. S'il tue une poule il perd 1 point, 3 points pour 1 chien, 5 points pour une vache et 10 points pour un ami. Le permis coûte 200 euros.

Écrire une fonction `amende` qui reçoit le nombre de victimes du chasseur et qui renvoie la somme due.

Utilisez cette fonction dans un programme principal qui saisit le nombre de victimes et qui affiche la somme que le chasseur doit déboursier.

10. Je suis ligoté sur les rails en gare d'Arras. Écrire un programme qui affiche un tableau me permettant de connaître l'heure à laquelle je serai déchiqueté par le train parti de la gare du Nord à 9h (il y a 170 km entre la gare du Nord et Arras).

Le tableau prédira les différentes heures possibles pour toutes les vitesses de 100 km/h à 300 km/h, par pas de 10 km/h, les résultats étant arrondis à la minute inférieure.

- Écrire une procédure `tchacatchac` qui reçoit la vitesse du train et qui affiche l'heure du drame ;
- écrire le programme principal qui affiche le tableau demandé.

▷ 11. Un programme principal saisit un *élément* d'ADN sous la forme d'une lettre minuscule qui ne peut être que "a", "t", "g" ou "c". Si l'élément est valide on le concatène à une chaîne (préalablement initialisée) nommée `adn`.

Écrire une fonction `valide` qui renvoie vrai si l'élément est valide, faux sinon.

Écrire une fonction `proportion` qui reçoit deux arguments, la chaîne `adn` et un codon `base` formé d'une suite d'éléments et qui retourne la proportion de `base` dans la chaîne.

Par ailleurs le programme principal saisit un codon (en vérifiant sa validité), appelle la fonction `proportion` et affiche le résultat.

12. Il s'agit d'écrire, d'une part, un programme principal, et d'autre part, une fonction utilisée dans le programme principal.

L'utilisateur remplit un tableau de  $N = 100$  entiers avec des entiers aléatoires en utilisant une fonction `randint(a, b)` qui retourne un entier entre  $a$  et  $b - 1$ . Une fonction nommée `indiceDuMin()` reçoit ce tableau et retourne l'indice de la case qui contient le minimum.

Écrire la fonction `indiceDuMin()`.

Écrire le programme qui échange le premier élément du tableau avec le minimum de ce tableau.

## Exercices

13. Un tableau *tab* comporte  $N = 100$  variables flottantes dont les  $n$  premières ( $n < 100$ ) sont utilisées.  
Écrire une fonction `indiceDuMax()` qui retourne l'indice du plus grand flottant parmi ces  $n$ , et une autre `indiceDuMin()` qui retourne l'indice du plus petit.  
Écrire ensuite un programme principal effectuant les actions suivantes :
  - saisie *filtrée* de  $n$  (vous devez faire en sorte que  $n$  ne puisse pas être saisi hors de ses limites) ;
  - remplissage aléatoire des  $n$  premières valeurs de *tab* (on utilisera le module `random()`, sans argument, qui retourne un flottant au hasard entre 0.0 et +1.0) ;
  - affichage de *l'amplitude* du tableau (écart entre sa plus grande et sa plus petite valeur) ;
  - affichage de la *moyenne* des  $n$  premières valeurs de *tab*.
  
14. Écrire une fonction `conv()` qui reçoit deux paramètres, une température et un entier  $n$ , et qui retourne la conversion Celsius  $\rightarrow$  Fahrenheit ( $n = 1$ ), ou Fahrenheit  $\rightarrow$  Celsius ( $n = 2$ ).  
Rappel :  $T_F = 32 + 1.8 \times T_C$
  
15. Fonction renvoyant plusieurs valeurs sous forme d'un *tuple*.  
Écrire une fonction `minMaxMoy` qui reçoit une liste d'entiers et qui renvoie le minimum, le maximum et la moyenne de cette liste. Le programme principal appellera cette fonction avec la liste : [10, 18, 14, 20, 12, 16].
  
16. Saisir un entier entre 1 et 3999 (pourquoi cette limitation ?). L'afficher en nombre romain.
  
17. Améliorer le script précédent en utilisant la fonction `zip()`. ◀
  
18. Un tableau contient  $n$  entiers ( $2 < n < 100$ ), tous compris entre 0 et 500. Vérifier qu'ils sont tous différents. ◀
  
19. L'utilisateur donne un entier  $n$  entre 2 et 12, le programme donne le nombre de façons de faire  $n$  en lançant deux dés.
  
20. Même problème que le précédent mais avec  $n$  entre 3 et 18 et trois dés.
  
21. Généralisation des deux questions précédentes. L'utilisateur saisit deux entrées, d'une part le nombre de dés, *nbd* (que l'on limitera pratiquement à 10), et d'autre part la somme,  $s$ , comprise entre *nbd* et  $6 \cdot nbd$ . Le programme calcule et affiche le nombre de façons de faire  $s$  avec les *nbd* dés. ◀◀
  
22. Même problème que le précédent mais codé récursivement. ◀◀

### *Exercices*

23. Pour faire des calculs sur les matrices carrées, on peut utiliser le type « liste de listes » et indexer un élément de la 3<sup>e</sup> ligne et 4<sup>e</sup> colonne par  $m[2][3]$  (compte-tenu que les indices commencent à 0).

On déclare trois matrices carrées de dimension  $N$  :  $m1$ ,  $m2$  et  $m3$  contenant des entiers.

On affecte  $m1$ , ligne à ligne, par les  $N^2$  premiers entiers pairs commençant à 2 ;  $m2$  est la matrice unité, c'est-à-dire qu'elle contient des 1 sur la diagonale principale (NW-SE) et des 0 partout ailleurs.

Pratiquement, on se limitera à  $Nmax = 10$ . Écrire l'algorithme du calcul de :

$$m3 = m1 - m2$$

.



## Solutions des exercices

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Volume d'un cône."""
__file__ = "exo_01.py"
__author__ = "Bob Cordeau"

# imports
from math import pi

# programme principal -----
rayon = input(u"Rayon du cône (m) : ")
hauteur = input(u"Hauteur du cône (m) : ")

volume = (pi*rayon*rayon*hauteur)/3.0
print "Volume du cône =", volume, "m3"
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Calcul d'un prix TTC."""
__file__ = "exo_02.py"
__author__ = "Bob Cordeau"

# programme principal -----
prixHT = float(raw_input("Prix HT (0 pour terminer)? "))
while prixHT > 0:
    print "Prix TTC : %.2f\n" % (prixHT * 1.196)
    prixHT = float(raw_input("Prix HT (0 pour terminer)? "))

print "\nAu revoir !"
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Somme d'entiers et nombre d'entiers supérieur à 100."""
__file__ = "exo_03.py"
__author__ = "Bob Cordeau"

# programme principal -----
somme, nombreTotal, nombreGrands = 0, 0, 0

x = int(raw_input("x (0 pour terminer) ? "))
while x > 0:
    somme = somme + x
    nombreTotal = nombreTotal + 1
    if x > 100:
        nombreGrands = nombreGrands + 1
    x = int(raw_input("x (0 pour terminer) ? "))

print "\nSomme :", somme
print nombreTotal, "valeur(s) en tout, dont", nombreGrands, "supérieure(s) à 100"
```

```
#!/bin/env python
```

## Solutions

```
# -*- coding: Latin-1 -*-
"""Parité."""
__file__ = "exo_04.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input("Entrez un entier strictement positif : ")
while n < 1:
    n = input("Entrez un entier STRICTEMENT POSITIF, s.v.p. : ")

if n%2 == 0:
    print n, "est pair."
else:
    print n, "est impair."
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Nobre de fois qu'un entier est divisible par 2."""
__file__ = "exo_05.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input("Entrez un entier strictement positif : ")
while n < 1:
    n = input("Entrez un entier STRICTEMENT POSITIF, s.v.p. : ")

cpt = 0
while n%2 == 0:
    n /= 2
    cpt += 1

print "Il est", cpt, "fois divisible par 2."
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Diviseurs propres d'un entier."""
__file__ = "exo_06.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input("Entrez un entier strictement positif : ")
while n < 1:
    n = input("Entrez un entier STRICTEMENT POSITIF, s.v.p. : ")

i = 2      # plus petit diviseur possible de n
cpt = 0    # initialise le compteur de divisions
p = n/2    # calculé une fois dans la boucle

print "Diviseurs propres sans répétition de ", n, ":",
while i <= p:
    if n%i == 0:
        cpt += 1
        print i,
    i += 1
```

## Solutions

```
if not cpt:
    print "aucun ! Il est premier."
else:
    print "(soit", cpt, "diviseurs propres)"
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Approximation de 'e'."""
__file__ = "exo_07.py"
__author__ = "Bob Cordeau"

# fonctions
def fact(n):
    r = 1
    for i in range(1, n+1):
        r *= i
    return r

# programme principal -----
n = int(raw_input("n ? "))
exp = 0.0
for i in range(n):
    exp = exp + 1.0/fact(i)

print "Approximation de 'e' : %.3f" % (exp)
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Gardien de phare."""
__file__ = "exo_08.py"
__author__ = "Bob Cordeau"

# fonctions
def hauteurparcourue(nb, h):
    print "Pour %d marches de %d cm, il parcourt %.2f m par semaine." \
          % (nb, h, nb*h*2*5*7/100.0)

# programme principal -----
nbMarches = int(raw_input("Combien de marches ? "))
hauteurMarche = int(raw_input("Hauteur d'une marche (cm) ? "))

hauteurparcourue(nbMarches, hauteurMarche)
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Permis de chasse."""
__file__ = "exo_09.py"
__author__ = "Bob Cordeau"

# fonctions
def permisSup(p, c, v, a):
    pointsPerdus = p + 3*c + 5*v + 10*a
    nbrePermis = pointsPerdus/100.0
```

## Solutions

```
    return 200*nbrePermis

# programme principal -----
poules = int(raw_input("Combien de poules ? "))
chiens = int(raw_input("Combien de chiens ? "))
vaches = int(raw_input("Combien de vaches ? "))
amis = int(raw_input("Combien d'amis ? "))

payer = permisSup(poules, chiens, vaches, amis)

print "\nA payer :",
if payer == 0:
    print "rien à payer"
else:
    print payer, "euros"

#!/bin/env python
# -*- coding: Latin-1 -*-
"""Histoire de train."""
__file__ = "exo_10.py"
__author__ = "Bob Cordeau"

# fonctions
def tchacatchac(v):
    "Affiche l'heure du drame"
    heure = 9 + int(170/v)
    minute = (60 * 170 / v) % 60
    print "A", v, "km/h, je me fais déchiqueter à", heure, "h", minute, "min."

# programme principal -----
i = 100
while i <= 300:
    tchacatchac(i)
    i += 10

#!/bin/env python
# -*- coding: Latin-1 -*-
"""Séquence d'ADN."""
__file__ = "exo_11.py"
__author__ = "Bob Cordeau"

# fonctions
def valide(c):
    "Retourne vrai si le chaînon est valide, faux sinon"
    if (c == 'a') or (c == 't') or (c == 'g') or (c == 'c'):
        return True
    else:
        return False

def proportion(c, b):
    "Retourne la proportion de la séquence <b> dans la chaîne <c>"
    n = len(c)
    k = c.count(b)
    return float(k)/n
```

## Solutions

```
# programme principal -----
print "\tSaisie de la chaîne d'ADN\n"
adn = ""
c = raw_input(u"chaînon : ")
if valide(c):
    adn = adn + c
while True:
    s = raw_input("Autre saisie (0/n)? ")
    if (s == 'n') or (s == 'N'):
        break
    else:
        c = raw_input(u"chaînon : ")
        if valide(c):
            adn = adn + c

print "\n\tSaisie de la séquence\n"
seq = ""
c = raw_input("chaînon : ")
if valide(c):
    seq = seq + c
while True:
    s = raw_input("Autre saisie (0/n)? ")
    if (s == 'n') or (s == 'N'):
        break
    else:
        c = raw_input("chaînon : ")
        if valide(c):
            seq = seq + c

# résultats
print '\nVotre chaîne : "%s"' %(adn)
print 'Il y a %.2f %% de "%s" dans votre chaîne.' % (proportion(adn, seq), seq)
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Echanges."""
__file__ = "exo_12.py"
__author__ = "Bob Cordeau"

# imports
from random import seed, randint

# fonctions
def listAleaInt(n, a, b):
    "Retourne une liste de <n> entiers aléatoires dans [<a> .. <b>]"
    return [randint(a, b) for i in xrange(n)]

# programme principal -----
seed() # initialise le générateur de nombres aléatoires
t = listAleaInt(100, 2, 125) # construction de la liste

# calcul de l'indice du minimum de la liste
iMin = t.index(min(t))
```

## Solutions

```
print "Avant échange :"  
print "\tt[0] =", t[0], "\tt[iMin] =", t[iMin]  
t[0], t[iMin] = t[iMin], t[0]  
print "Après échange :"  
print "\tt[0] =", t[0], "\tt[iMin] =", t[iMin]
```

```
#!/bin/env python  
# -*- coding: Latin-1 -*-  
"""Amplitude et moyenne d'une liste de flottants."""  
__file__ = "exo_13.py"  
__author__ = "Bob Cordeau"
```

```
# imports  
from random import seed, random
```

```
# fonctions  
def listAleaFloat(n):  
    "Retourne une liste de <n> flottants aléatoires"  
    return [random() for i in xrange(n)]
```

```
# programme principal -----
```

```
n = input("Entrez un entier [2 .. 100] : ")  
while n < 2 or n > 100: # saisie filtrée  
    n = input("Entrez un entier [2 .. 100], s.v.p. : ")
```

```
seed() # initialise le générateur de nombres aléatoires  
t = listAleaFloat(n) # construction de la liste
```

```
print "Amplitude : %.2f" % (max(t) - min(t))  
print "Moyenne : %.2f" % (sum(t)/n)
```

```
#!/bin/env python  
# -*- coding: Latin-1 -*-  
"""Conversions de températures."""  
__file__ = "exo_14.py"  
__author__ = "Bob Cordeau"
```

```
# fonctions  
def conv(t, n):  
    "Renvoie la conversion Celsius -> Fahrenheit ou inversement suivant <n>"  
    if n == 1: # Celsius -> Fahrenheit  
        return (32.0 + 1.8*t)  
    elif n == 2: # Fahrenheit -> Celsius  
        return ((t-32.0)/1.8)
```

```
# programme principal -----
```

```
t = float(raw_input(u"Température ? "))  
u = float(raw_input(u"Unité (1 = Celsius, 2 = Fahrenheit) ? "))  
while (u < 1) or (u > 2):  
    u = float(raw_input(u"Unité (1 = Celsius, 2 = Fahrenheit), SVP ? "))
```

```
unit = {1: '°C', 2: '°F'}  
print "\n%.2f %s = %.2f %s" % (t, unit[u], conv(t, u), unit[(u%2)+1])
```

## Solutions

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Min, max et moyenne d'une liste d'entiers."""
__file__ = "exo_15.py"
__author__ = "Bob Cordeau"

# fonctions
def minMaxMoy(liste):
    "Renvoie le min, le max et la moyenne de la liste"
    n = len(liste)
    if n == 0:
        return None
    min = max = som = liste[0]
    for i in liste[1:]:
        if i < min:
            min = i
        if i > max:
            max = i
        som = som + i
    return (min, max, som/float(n))

# programme principal -----
lp = [10, 18, 14, 20, 12, 16]

print "min : %d, max : %d, moy : %.2f" % (minMaxMoy(lp))
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Nombres romains (1)."""
__file__ = "exo_16.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input('Entrez un entier [1 .. 4000[ : ')
while (n < 1 or n > 3999):
    n = input('Entrez un entier [1 .. 4000[, s.v.p. : ')

s = "" # Chaîne résultante

while n >= 1000:
    s += "M"
    n -= 1000

if n >= 900:
    s += "CM"
    n -= 900

if n >= 500:
    s += "D"
    n -= 500

if n >= 400:
    s += "CD"
```

## Solutions

```
n -= 400

while n >= 100:
    s += "C"
    n -= 100

if n >= 90:
    s += "XC"
    n -= 90

if n >= 50:
    s += "L"
    n -= 50

if n >= 40:
    s += "XL"
    n -= 40

while n >= 10:
    s += "X"
    n -= 10

if n >= 9:
    s += "IX"
    n -= 9

if n >= 5:
    s += "V"
    n -= 5

if n >= 4:
    s += "IV"
    n -= 4

while n >= 1:
    s += "I"
    n -= 1

print "En romain :", s
```

---

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Nombres romains (2)."""
__file__ = "exo_17.py"
__author__ = "Bob Cordeau"

# globales
code = zip(
    [1000,900 ,500,400 ,100,90 ,50 ,40 ,10 ,9 ,5 ,4 ,1],
    ["M" , "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"]
)

# fonctions
def decToRoman(num):
```

## Solutions

```
res = []
for d, r in code:
    while num >= d:
        res.append(r)
        num -= d
    return ''.join(res)

# programme principal -----
for i in xrange(1,4000):
    print i, decToRoman(i)
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Liste d'entiers différents."""
__file__ = "exo_18.py"
__author__ = "Bob Cordeau"

# imports
from random import seed, randint

# fonctions
def listAleaInt(n, a, b):
    "Retourne une liste de <n> entiers aléatoires entre <a> et <b>"
    return [randint(a, b) for i in xrange(n)]

# programme principal -----
N = 100
n = input("Entrez un entier [1 .. 100] : ")
while n < 1 or n > N:
    n = input("Entrez un entier [1 .. 100], s.v.p. : ")

# construction de la liste
seed() # initialise le générateur de nombres aléatoires
t = listAleaInt(n, 0, 500)

# Sont-ils différents ?
tousDiff = True
i = 0
while tousDiff and i < (n-1):
    j = i + 1
    while tousDiff and j < n:
        if t[i] == t[j]:
            tousDiff = False
        else:
            j += 1
    i += 1

print "\n", t,
if tousDiff:
    print ": tous les éléments sont distincts."
else:
    print ": au moins une valeur est répétée."
```

```
#!/bin/env python
```

## Solutions

```
# -*- coding: Latin-1 -*-
"""Jeu de dés (1)."""
__file__ = "exo_19.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input("Entrez un entier [2 .. 12] : ")
while n < 2 or n > 12:
    n = input("Entrez un entier [2 .. 12], s.v.p. : ")

s = 0
for i in xrange(1, 7):
    for j in xrange(1, 7):
        if i+j == n:
            s += 1

print "Il y a", s, "façon(s) de faire", n, " avec deux dés."
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Jeu de dés (2)."""
__file__ = "exo_20.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input("Entrez un entier [3 .. 18] : ")
while n < 3 or n > 18:
    n = input("Entrez un entier [3 .. 18], s.v.p. : ")

s = 0
for i in xrange(1, 7):
    for j in xrange(1, 7):
        for k in xrange(1, 7):
            if i+j+k == n:
                s += 1

print "Il y a", s, "façon(s) de faire", n, " avec trois dés."
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Jeu de dés (3)."""
__file__ = "exo_21.py"
__author__ = "Bob Cordeau"

# globales
MAX = 8

# programme principal -----
nbd = input(u"Nombre de dés entre 2 et %d :" % MAX)
while nbd < 2 or nbd > MAX:
    nbd = input(u"Nombre de dés entre 2 et %d, s.v.p. :" % MAX)

s = input("Entrez un entier entre %d et %d :" % (nbd, 6*nbd,))
while s < nbd or s > 6*nbd:
```

## Solutions

```
s = input("Entrez un entier entre", nbd, "et", 6*nbd, ", s.v.p. :")

if s == nbd or s == 6*nbd:
    cpt = 1 # 1 seule solution
else:
    I = [1]*nbd # initialise une liste de <nbd> dés
    cpt, j = 0, 0
    while j < nbd:
        som = sum([I[k] for k in xrange(nbd)])

        if som == s:
            cpt += 1 # compteur de bonnes solutions
        if som == 6*nbd:
            break

        j = 0
        if I[j] < 6:
            I[j] += 1
        else:
            while I[j] == 6:
                I[j] = 1
                j += 1
            I[j] += 1

print "Il y a", cpt, "façons de faire", s, "avec", nbd, "dés."
```

```
#!/bin/env python
# -*- coding: Latin-1 -*-
"""Jeu de dés (récursif)."""
__file__ = "exo_22.py"
__author__ = "Bob Cordeau"

# imports
from easygui import integerbox, msgbox

# globales
MAX = 8

# fonctions
def calcul(d, n):
    "Calcul récursif du nombre de façons de faire <n> avec <d> dés"
    resultat, debut = 0, 1
    if d == 1 or n == d or n == 6*d: # conditions terminales
        return 1
    else: # sinon appels récursifs
        if n > 6*(d-1): # optimisation importante
            debut = n - 6*(d-1)

        for i in xrange(debut, 7):
            if n == i:
                break
            resultat += calcul(d-1, n-i)
    return resultat
```

## Solutions

```
# programme principal -----
d = integerbox("nombre de dés [2 .. %d]" % MAX, "Saisie du nombre de dés", 5, 2, MAX)
n = integerbox("Entrez un entier [%d .. %d]" % (d, 6*d), "nombre", 5*d/2, d, 6*d)
msgbox("Il y a %d façon(s) de faire %d avec %d dés." % (calcul(d, n), n, d))

#!/bin/env python
# -*- coding: Latin-1 -*-
"""Calculs de matrices."""
__file__ = "exo_23.py"
__author__ = "Bob Cordeau"

# programme principal -----
n = input("Dimension des matrices carrées [2 .. 10] : ")

# initialisation des 3 matrices
m1, m2, m3 = [0]*n, [0]*n, [0]*n
for i in xrange(n):
    m1[i], m2[i], m3[i] = [0]*n, [0]*n, [0]*n

# calcul des matrices
k = 2
for i in xrange(n):
    for j in xrange(n):
        m1[i][j] = k # matrice d'éléments pairs
        k += 2

        if i == j:
            m2[i][j] = 1 # matrice unité

        m3[i][j] = m1[i][j] - m2[i][j]

# Affichages
print "\n m1 :"
for i in xrange(n):
    print m1[i]
print "\n m2 :"
for i in xrange(n):
    print m2[i]
print "\n m3 = m1 - m2 :"
for i in xrange(n):
    print m3[i]
```



# Colophon

Ces exercices ont été composés grâce au logiciel  $\text{\LaTeX}$  sous l'éditeur  $\text{\TeXnicCenter}$ . Le corps du texte est en police **Utopia**, les exemples de code en police Typewriter.

Ce document est disponible à l'adresse :

[www.iut-orsay.fr/dptmphy/Pedagogie/Welcome.html](http://www.iut-orsay.fr/dptmphy/Pedagogie/Welcome.html)

