

## Préparation aux oraux de la banque PT Informatique

### Exercices

## Préparation aux oraux de la banque PT Épreuve de « Mathématique et Algorithmique »

Exercice 1 – Arithmétique	2
Exercice 2 – Intégration	2
Exercice 3 – Graphe	2
Exercice 4 – Gestion de liste	2
Exercice 5 – Probabilités	3
Exercice 6 – Tracer de fonction – $f(x)=0$	3
Exercice 7 – Algorithmique	3
Exercice 8 – Chiffrer – déchiffrer	3
Exercice 9 – Fractale de Mandelbrot	3
Exercice 10 – Calcul matriciel	4
Exercice 11 – Tri de liste	4
Exercice 12 – Courbes paramétrées	4
Exercice 1 – Arithmétique – Corrigé	5
Exercice 2 – Intégration – Corrigé	5
Exercice 3 – Graphe – Corrigé	6
Exercice 5 – Corrigé	7
Exercice 6 – Corrigé	7
Exercice 7 – Corrigé	8
Exercice 8 – Corrigé	8
Exercice 9 – Fractale de Mandelbrot – Corrigé	9
Exercice 10 – Corrigé	10
Exercice 11 – Tri de liste – Corrigé	11
Exercice 12 – Corrigé	11

### Exercice 1 – Arithmétique

1. Soit l'entier  $n = 1234$ . Quel est le quotient, noté  $q$ , dans la division euclidienne de  $n$  par 10 ? Quel est le reste ? Que se passe-t-il si on recommence la division par 10 à partir de  $q$  ?
2. Écrire la suite d'instructions calculant la somme des cubes des chiffres de l'entier 1234.
3. Écrire une fonction `somcube`, d'argument  $n$ , renvoyant la somme des cubes des chiffres du nombre entier  $n$ .
4. Trouver tous les nombres entiers inférieurs à 1000 égaux à la somme des cubes de leurs chiffres.
5. En modifiant les instructions de la fonction `somcube`, écrire une fonction `somcube2` qui convertit l'entier  $n$  en une chaîne de caractères permettant ainsi la récupération de ses chiffres sous forme de caractères. Cette nouvelle fonction renvoie toujours la somme des cubes des chiffres de l'entier  $n$ .

### Exercice 2 – Intégration

On cherche à calculer une valeur approchée de l'intégrale d'une fonction donnée par des points dont les coordonnées sont situées dans un fichier.

1. Le fichier `ex_01.txt`, situé dans le sous-répertoire `data` du répertoire de travail, contient une quinzaine de lignes selon le modèle suivant :

```
0.0;1.00988282142
0.1;1.07221264497
```

Chaque ligne contient deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point. Les points sont ordonnés par abscisses croissantes. Ouvrir le fichier en lecture, le lire et construire la liste `LX` des abscisses et la liste `LY` des ordonnées contenues dans ce fichier.

2. Représenter les points sur une figure.
3. Les points précédents sont situés sur la courbe représentative d'une fonction  $f$ . On souhaite déterminer une valeur approchée de l'intégrale  $I$  de cette fonction sur le segment où elle est définie. Écrire une fonction `trapeze`, d'arguments deux listes  $y$  et  $x$  de même longueur  $n$ , renvoyant :

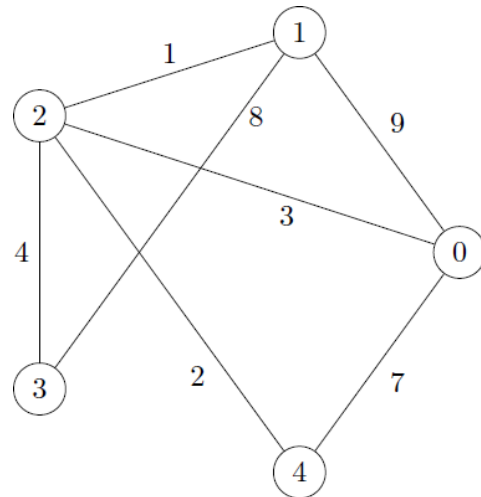
$$\sum_{i=1}^{n-1} (x_i - x_{i-1}) \frac{y_i + y_{i-1}}{2}.$$

`trapeze` (`LY`, `LX`) renvoie donc une valeur approchée de l'intégrale  $I$  par la méthode des trapèzes.

4. En utilisant la méthode d'intégration numérique `trapz` de la sous-bibliothèque `scipy.integrate` du langage Python ou la méthode `inttrap` du logiciel Scilab, retrouver la valeur approchée de l'intégrale  $I$ .

### Exercice 3

On considère le graphe  $G$  suivant, où le nombre situé sur l'arête joignant deux sommets est leur distance, supposée entière :



1. Construire la matrice  $(M_{ij})_{0 \leq i,j \leq 4}$ , matrice de distances du graphe  $G$ , définie par : « pour tous les indices  $i, j$ ,  $M_{ij}$  représente la distance entre les sommets  $i$  et  $j$ , ou encore la longueur de l'arête reliant les sommets  $i$  et  $j$  ». On convient que, lorsque les sommets ne sont pas reliés, cette distance vaut -1. La distance du sommet  $i$  à lui-même est, bien sûr, égale à 0.
2. Écrire une suite d'instructions permettant de dresser à partir de la matrice  $M$  la liste des voisins du sommet 4.
3. Écrire une fonction `voisins`, d'argument un sommet  $i$ , renvoyant la liste des voisins du sommet  $i$ .
4. Écrire une fonction `degre`, d'argument un sommet  $i$ , renvoyant le nombre des voisins du sommet  $i$ , c'est-à-dire le nombre d'arêtes issues de  $i$ .
5. Écrire une fonction `longueur`, d'argument une liste  $L$  de sommets de  $G$ , renvoyant la longueur du trajet d'écrit par cette liste  $L$ , c'est-à-dire la somme des longueurs des arêtes empruntées. Si le trajet n'est pas possible, la fonction renverra -1.

### Exercice 4 – Gestion de liste

Soit un entier naturel  $n$  non nul et une liste  $t$  de longueur  $n$  dont les termes valent 0 ou 1. Le but de cet exercice est de trouver le nombre maximal de 0 contigus dans  $t$  (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de zéros contigus de la liste  $t_1$  suivante vaut 4 :

$i$	0	1	2	3	4	5	6	7
$t_1[i]$	0	1	1	1	0	0	0	1

$i$	8	9	10	11	12	13	14
$t_1[i]$	0	1	1	0	0	0	0

1. Écrire une fonction `nombreZeros` ( $t, i$ ), prenant en paramètres une liste  $t$ , de longueur  $n$ , et un indice  $i$  compris entre 0 et  $n - 1$ , et renvoyant :

$$\begin{cases} 0, & \text{si } t[i] = 1 \\ \text{le nombre de zéros consécutifs dans } t \\ \text{à partir de } t[i] \text{ inclus, si } t[i] = 0. \end{cases}$$

Par exemple, les appels `nombreZeros(t1,4)`, `nombreZeros(t1,1)` et `nombreZeros(t1,8)` renvoient respectivement les valeurs 3, 0 et 1.

- Comment obtenir le nombre maximal de zéros contigus d'une liste `t` connaissant la liste des `nombreZeros(t,i)` pour  $0 \leq i \leq n-1$ ? En déduire une fonction `nombreZerosMax(t)`, de paramètre `t`, renvoyant le nombre maximal de 0 contigus d'une liste `t` non vide. On utilisera la fonction `nombreZeros`.
- Quelle est la complexité de la fonction `nombreZerosMax(t)` construite à la question précédente?
- Trouver un moyen simple, toujours en utilisant la fonction `nombreZeros`, d'obtenir un algorithme plus performant.

### Exercice 5 – Probabilités

Soient  $n$  un entier naturel strictement positif et  $p$  un réel compris entre 0 et 1. On considère  $X$  et  $Y$  deux variables aléatoires à valeurs dans  $\mathbb{N}$  sur un espace probabilisé donné.  $X$  suit une loi de Poisson de paramètre  $\lambda = np$  et  $Y$  suit une loi binomiale de paramètres  $(n, p)$ .

- Définir une fonction `Px`, d'arguments  $k$ ,  $n$  et  $p$ , renvoyant la valeur de  $P(X = k)$ .  $k!$  (factorielle  $k$ ) s'obtient par `factorial(k)` en Python (bibliothèque `math`) et `prod(1 : k)` en Scilab. Déterminer, pour  $n = 30$  et  $p = 0,1$ , la liste des valeurs de  $P(X = k)$  pour  $k \in \mathbb{N}$ ,  $0 \leq k \leq 30$ .
- Définir une fonction `Py`, d'arguments  $k$ ,  $n$  et  $p$ , renvoyant la valeur de  $P(Y = k)$ . On pourra utiliser `comb` de la sous-bibliothèque `scipy.misc` en Python et `binomial` en Scilab. Déterminer, pour  $n = 30$  et  $p = 0,1$ , la liste des valeurs de  $P(Y = k)$  pour  $k \in \mathbb{N}$ ,  $0 \leq k \leq 30$ .
- Soit  $k \in \mathbb{N}$ . On rappelle que, sous certaines conditions sur  $n$  et  $p$ , la probabilité  $P(Y = k)$  peut être approchée par  $P(X = k)$ . Déterminer une fonction `Ecart` d'arguments  $n$  et  $p$ , renvoyant le plus grand des nombres  $|P(Y = k) - P(X = k)|$ , pour  $0 \leq k \leq n$ .
- Soit  $e$  un réel strictement positif. Déterminer une fonction `N`, d'arguments  $e$  et  $p$ , renvoyant le plus petit entier  $n$  tel que `Ecart(n, p)` soit inférieur ou égal à  $e$ .
- Faire l'application numérique dans les quatre cas suivants :
  - $p = 0,075$  avec  $e = 0,008$  et  $e = 0,005$ ;
  - $p = 0,1$  avec  $e = 0,008$  et  $e = 0,005$ . Interpréter le dernier résultat.

### Exercice 6 – $f(x) = 0$

On considère la fonction  $g$  définie sur  $[0,2[$  par :

$$g(x) = \begin{cases} x & \text{pour } 0 \leq x < 1 \\ 1 & \text{pour } 1 \leq x < 2 \end{cases}$$

- Définir la fonction  $g$ . Tracer sa courbe représentative sur  $[0,2[$ , c'est-à-dire la ligne brisée reliant les points  $(x, g(x))$  pour  $x$  variant de 0 à 1,99 avec un pas de 0,01.

- Définir une fonction  $f$  donnée de manière récursive sur  $[0, +\infty[$  par :

$$f(x) = \begin{cases} g(x) & \text{pour } 0 \leq x < 2 \\ \sqrt{x} f(x-2) & \text{pour } x \geq 2 \end{cases}$$

- Tracer la courbe représentative de  $f$  sur  $[0,6]$ .
- Écrire les instructions permettant de calculer, à  $10^{-2}$  près, la plus petite valeur  $\alpha > 0$  telle que  $f(\alpha) > 4$ .

### Exercice 7 – Algorithmique

On considère le code Python de la fonction `d` suivante :

#### ■ Python

```
def d(n):
    L = [1]
    for nombre in range(2,n+1):
        if n%nombre == 0:
            L.append(nombre)
    return L
```

- Quel est le résultat de l'appel `d(4)` ? Puis de l'appel `d(10)` ? Que fait la fonction `d` ?
- Un diviseur non-trivial d'un entier  $n$  est un diviseur de  $n$  différent de 1 et de  $n$ . Écrire une fonction `DNT`, d'argument  $n$ , renvoyant la liste des diviseurs non-triviaux de l'entier  $n$ .
- Écrire une fonction `sommeCarresDNT`, d'argument  $n$ , renvoyant la somme des carrés des diviseurs non-triviaux de l'entier  $n$ .
- Écrire la suite des instructions permettant d'afficher tous les nombres entiers inférieurs à 1000 et égaux à la somme des carrés de leurs diviseurs non-triviaux. Que peut-on conjecturer ?

### Exercice 8 – Chiffrer – déchiffrer

Soit  $n$  un entier vérifiant  $n \leq 26$ . On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de  $n$  lettres. Par exemple pour  $n = 3$ , le décalage sera le suivant :

Avant décalage	a	b	c	...	x	y	z
Après décalage	d	e	f	...	a	b	c

Le mot `oralensam` devient ainsi `rudohqvdp`.

- Définir une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique (caractères en minuscule).
- Écrire une fonction `decalage`, d'argument un entier  $n$ , renvoyant une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique, décalées de  $n$ , comme indiqué ci-dessus.
- Écrire une fonction `indices`, d'arguments un caractère  $x$  et une chaîne de caractères `phrase`, renvoyant une liste contenant les indices de  $x$  dans `phrase` si  $x$  est une lettre de phrase et une liste vide sinon.
- Écrire une fonction `codage` d'arguments un entier  $n$  et une chaîne de caractères `phrase`, renvoyant `phrase` codé avec un décalage de  $n$  lettres.
- Comment peut-on décoder un mot codé ?

### Exercice 9 – Fractale de Mandelbrot

On pose  $M = 20$  et  $m = 10$ . À un nombre  $c$  quelconque, on associe la suite  $(u_n)_{n \geq 0}$  définie par  $u_0 = 0$  et  $u_{n+1} = u_n^2 + c$  pour  $n \geq 0$ .

S'il existe, on note  $k$  le plus petit entier tel que l'on ait  $0 \leq k \leq m$  et  $|u_k| > M$ . On définit alors la fonction  $f$  par

$$f : c \mapsto \begin{cases} k & \text{s'il existe} \\ m + 1 & \text{sinon.} \end{cases}$$

1. Donner le code définissant la fonction  $f$ .
2. Tracer l'allure de la courbe représentative de la fonction  $f$  sur  $[-2; 2]$ , en créant une liste LX de 401 valeurs équiréparties entre -2 et 2 inclus et en utilisant les fonctions `plot` et `show` de la sous-bibliothèque `matplotlib.pyplot`.
3. Construire le tableau des valeurs  $f(x + iy)$  où  $x$  prend 101 valeurs comprises entre -2 et 0,5 et  $y$  prend 101 valeurs entre -1,1 et 1,1. On rappelle que le nombre complexe  $i$  est représenté par `1j`. Par exemple, le complexe  $1 + 2i$  est représenté par `1 + 2j`.
4. Tracer l'image que code ce tableau. On pourra utiliser les fonctions `imshow` et `show` de la sous-bibliothèque `matplotlib.pyplot`. Quels paramètres peut-on modifier pour obtenir une meilleure résolution?

### Exercice 10 – Calcul matriciel

Dans cet exercice, avec Python on pourra utiliser la fonction `array` de la bibliothèque `numpy`, ainsi que la fonction `eig` de la sous-bibliothèque `numpy.linalg`. Avec Scilab, on utilisera `spec`.

1. Créer deux matrices  $R = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$  et  $S = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$  et les faire afficher.
2. Créer une fonction `test`, d'argument  $M$ , renvoyant la valeur `n` si  $M$  est une matrice carrée d'ordre `n` (entier naturel non nul) et zéro dans tous les autres cas. Vérifier la fonction `test` sur  $R$  et sur  $S$ .
3. Le fichier `ex_006.txt`, situé dans le sous-répertoire `data` du répertoire de travail, contient un tableau de valeurs flottantes. Lire ce tableau dans le fichier et vérifier qu'il correspond bien à une matrice carrée d'ordre 5 que l'on désignera par  $M1$ .
4. Déterminer les valeurs propres de la matrice  $M1$ .
5. Créer une fonction `dansIntervalle`, d'arguments une liste  $L$  et deux réels  $a$  et  $b$ , renvoyant la valeur `True` si tous les éléments de la liste  $L$  sont dans l'intervalle  $[a, b]$  et `False` sinon. Vérifier que toutes les valeurs propres de la matrice  $M1$  sont dans l'intervalle  $[0, 1]$ .

### Exercice 11 – Tri de liste

Soit  $N$  un entier naturel non nul. On cherche à trier une liste  $L$  d'entiers naturels strictement inférieurs à  $N$ .

1. Écrire une fonction `comptage`, d'arguments  $L$  et  $N$ , renvoyant une liste  $P$  dont le  $k$ -ième élément dé-

signe le nombre d'occurrences de l'entier  $k$  dans la liste  $L$ .

2. Utiliser la liste  $P$  pour en déduire une fonction `tri`, d'arguments  $L$  et  $N$ , renvoyant la liste  $L$  triée dans l'ordre croissant.
3. Tester la fonction `tri` sur une liste de 20 entiers inférieurs ou égaux à 5, tirés aléatoirement.
4. Quelle est la complexité temporelle de cet algorithme? La comparer à la complexité d'un tri par insertion ou d'un tri fusion.

### Exercice 12 – Courbes paramétrées

1. Deux paramètres  $b$  et  $w$  valant respectivement 0,5 et 6,0, définir trois fonctions d'une variable  $t$  renvoyant des couples :

$$\begin{cases} p : t \mapsto (\cos(t) + b \cos(wt), \sin(t) + b \sin(wt)) \\ v : t \mapsto (-\sin(t) - bw \sin(wt), \cos(t) + bw \cos(wt)) \\ a : t \mapsto (-\cos(t) - bw^2 \cos(wt), -\sin(t) - bw^2 \sin(wt)) \end{cases}$$

Vérifier ces fonctions sur un exemple.

$p(t) = (x(t), y(t))$  désigne la position dans le plan d'une masse ponctuelle mobile au cours du temps,  $v(t) = (x'(t), y'(t))$ , sa vitesse, et  $a(t) = (x''(t), y''(t))$ , son accélération.

2. Construire la liste  $L$  des points  $p(t)$ , pour  $t$  variant de  $-\pi$  à  $\pi$  avec un pas de discrétisation  $\delta t$  vérifiant  $\delta t = 0,01 \pi$ .
3. Faire tracer dans le plan muni d'un repère orthonormal la ligne polygonale reliant les points  $p(t)$  de la liste  $L$ .
4. Définir puis tester la fonction  $c$  d'une variable  $t$  qui renvoie le couple des coordonnées du centre de courbure donnée par :

$$c(t) = (x(t) - dy'(t), y(t) + dx'(t))$$

où

$$d = \frac{x'(t)^2 + y'(t)^2}{x'(t)y''(t) - y'(t)x''(t)}.$$

5. Rajouter sur le graphique précédent la ligne décrite par les centres de courbure, avec la même discrétisation en temps.
6. Calculer la longueur de la ligne polygonale reliant les points  $p(t)$ , pour différents pas de discrétisation  $\delta t$ . Observer l'évolution de cette longueur lorsque  $\delta t$  diminue.

## Exercice 1 – Arithmétique – Corrigé

```
# Question 1
n = 1234
q = n//10
r = n%q

# r contient le nombre d'unités de n
```

```
# Question 2
s=0
while n!=0:
    q=n//10
    r = n%10
    #print(r)
    s=s+r**3
    n=q
```

```
# Question 3
def somcube(n):
    """
    Entrées :
    * n, int : nombre
    Sortie :
    * s, int : somme des cubes du chiffre n
    """
    s=0
    while n!=0:
        q=n//10
        r = n%10
        s=s+r**3
        n=q
    return s
```

```
# Question 4
res = []
for i in range(10001):
    if i == somcube(i):
        res.append(i)
```

```
# Question 5
def somcube2(n):
    """
    Entrées :
    * n, int : nombre
    Sortie :
    * s, int : somme des cubes du chiffre n
    """
    nombre=str(n)
    s=0
    for chiffre in nombre :
        s = s+int(chiffre)**3
    return s

print(somcube2(1234))
```

## Exercice 2 – Intégration – Corrigé

```
# Question 1
# =====
# Le répertoire courant est Exercice_02.
# Le sous-répertoire data contient le
# fichier ex_01.txt.

# On ouvre le fichier en lecture)
fid = open("data\ex_01.txt")

# On charge le fichier dans une liste.
# Chaque élément de la liste correspond à
# chaque ligne sous forme de chaîne de caractère.
file = fid.readlines()
# On ferme le fichier
fid.close()
```

```
LX=[]
LY=[]
for ligne in file :
    ligne = ligne.split(';')
    LX.append(float(ligne[0]))
    LY.append(float(ligne[1]))
```

```
# Question 2
# =====
# Ne pas oublier de charger préalablement
# import matplotlib.pyplot as plt
```

```
plt.plot(LX,LY)
plt.show()
```

```
# Question 3
# =====
def trapeze(x,y):
    res = 0
    for i in range(1,len(LX)):
        res = res+(LX[i]-LX[i-1])*0.5*(LY[i]+LY[i-1])
    return res
print(trapeze(LX,LY))
```

```
# Question 4
# =====
from scipy.integrate import trapz
# Attention à l'ordre des arguments dans
# la fonction trapz : les_y puis les_x
# Après l'import, help(trapz) permet d'avoir
# de l'aide sur la fonction.
print(trapz(LY,LX))
```

## Exercice 3 – Graphe – Corrigé

```
# Question 1
# =====
# Matrices avec des listes
M=[[0,9,3,-1,7],
   [9,0,1,8,-1],
   [3,1,0,4,2],
   [-1,8,4,0,-1],
   [7,-1,2,-1,0]]
```

```
# Question 2 & 3
# =====
def voisins(M,i):
    """
    Entrées :
    * M(lst) : graphe
    * i : noeud considéré
    Sortie :
    * v(lst) : liste des voisins
    """
    v = []
    # On cherche les voisins sur une ligne
    # (on pourrait le faire sur une colonne)
    for j in range(len(M[i])):
        if M[i][j]>0:
            v.append(j)
    return v

# print(voisins(M,0))
```

```
# Question 4
# =====
def degre(M,i):
    """
    Entrées :
    * M(lst) : graphe
    * i : noeud considéré
    Sortie :
    * (int) : nombre de voisins
    """
    return len(voisins(M,i))
```

```
# Question 5
# =====
def longueur(M,chemin):
    l = 0
    for i in range(len(chemin)-1):
        if M[chemin[i]][chemin[i+1]]<0:
            return -1
        else :
            l=l+M[chemin[i]][chemin[i+1]]
    return l

chemin = [1,2,3,1,4]
print(longueur(M,chemin))
chemin = [0,4,2,1,0]
print(longueur(M,chemin))
```

## Exercice 4

```
# Question 1
# =====
def nombreZeros(t,i):
    if t[i]==1:
        return 0
    else :
        res = 1
        j=i+1
        while j<len(t) and t[j]==0:
            res = res+1
            j=j+1
        return res
# t1=[0,1,1,1,0,0,0,1,0,1,1,0,0,0,0]
# print(nombreZeros(t1,4))
# print(nombreZeros(t1,1))
# print(nombreZeros(t1,8))
```

```
# Question 2
# =====
def nombreZerosMax(t):
    max=nombreZeros(t,0)
    for i in range(1,len(t)):
        tmp = nombreZeros(t,i)
        if tmp>max:
            max = tmp
    return max
print(nombreZerosMax(t1))
```

```
# Question 3 et 4
# =====
```

## Exercice 5 – Corrigé

## Exercice 6 – Corrigé

```
# Import de fonctions
import matplotlib.pyplot as plt
from math import sqrt
# Question 1
# =====
def g(x):
    if x >= 0 and x < 1 :
        return x
    elif x > 1 and x < 2 :
        return 1
xx = [0]
t = 0
while t <= 1.99:
    t = t + 0.01
    xx.append(t)
yy = [g(x) for x in xx]
plt.plot(xx, yy)
plt.show()
```

```
# Question 2
# =====
def f(x):
    if x >= 0 and x < 2 :
        return g(x)
    else : # x >= 2
        return sqrt(x) * f(x - 2)
```

```
# Question 3
# =====
xxx = [0]
t = 0
while t <= 6:
    t = t + 0.01
    xxx.append(t)
yyy = [f(x) for x in xxx]
plt.plot(xxx, yyy)
plt.show()
```

```
# Question 4
# =====
# On cherche à résoudre  $f(x) - 4 = 0$  sur
# l'intervalle [5,6]
def h(x):
    res = f(x) - 4
    return res

a = 5.
b = 6.
while (b - a) > 0.01:
    m = (a + b) / 2
    if h(m) > 0:
        b = m
    else :
        a = m
m = (a + b) / 2

if h(m) < 0:
    m = m + abs(b - a)
print(m, h(m))
```



## Exercice 7 – Corrigé

```
# Question 1
# =====
def d(n):
    """
    Retourne la liste de tous les diviseurs de n.
    Entrée :
    * n(int) : entier.
    Sortie :
    * L(lst) : liste des diviseurs de n.
    """
    L = [1]
    for nombre in range(2, n+1):
        if n%nombre == 0:
            L.append(nombre)
    return L
print(d(4), d(10))
```

```
# Question 2
# =====
def DNT_01(n):
    return d(n)[1:-1]
def DNT_02(n):
    L = []
    for nombre in range(2, n):
        if n%nombre == 0:
            L.append(nombre)
    return L
print(DNT_01(4), DNT_02(4))
print(DNT_01(10), DNT_02(10))
```

```
# Question 3
# =====
def sommeCarresDNT_01(n):
    L = DNT_01(n)
    res = [x**2 for x in L]
    return sum(res)
def sommeCarresDNT_02(n):
    L = DNT_01(n)
    res = 0
    for x in L:
        res = res + x*x
    return res
def sommeCarresDNT_03(n):
    L = DNT_01(n)
    res = 0
    for i in range(len(L)):
        res = res + L[i]**2
    return res
print(sommeCarresDNT_01(15), sommeCarresDNT_02(15),
      sommeCarresDNT_03(15))
```

```
# Question 4
# =====
from math import sqrt
for i in range(1001):
    if i == sommeCarresDNT_01(i):
        print(str(i)+"\t"+str(sqrt(i)))
# Conjecture les nombres recherchés sont
# les carrés des nombres premiers.
```

## Exercice 8 – Corrigé

```
# Question 1
# =====
chaine = "abcdefghijklmnopqrstuvwxyz"
```

```
# Question 2
# =====
def decalage(chaine, n):
    chaine = chaine[n:-1] + chaine[0:n]
    return chaine
print(chaine, decalage(chaine, 3))
```

```
# Question 3
# =====
def indices(x, phrase):
    """
    Recherche des indices de x dans phrase
    Entrée :
    * x(str) : un caractère
    * phrase(str)
    Sortie :
    * res(lst) : liste des indices de x
    """
    res = []
    for i in range(len(phrase)):
        if phrase[i] == x:
            res.append(i)
    return res
print(indices("a", "akjlkjalkjlkjalkjlkja"))
```

```
# Question 4
# =====
def codage(n, phrase):
    ch = "abcdefghijklmnopqrstuvwxyz"
    ch_c = decalage(ch, n)
    print(ch_c)
    phrase_c = ""
    for c in phrase:
        i = indices(c, ch)
        i = i[0]
        phrase_c = phrase_c + ch_c[i]
    return phrase_c
print(codage(3, "oralensam"))
```

```
# Question 5
# =====
# Solution 1 : essayer les 26 permutations,
# jusqu'à trouver une phrase qui est du sens.
# Solution 2 : statistiquement le e est la lettre
# la plus présente dans la langue française. On
# peut donc déterminer la fréquence d'apparition
# des lettres. # La lettre la plus fréquente
# peut être assimilée au "e".
# On calcule ainsi le décalage...
```



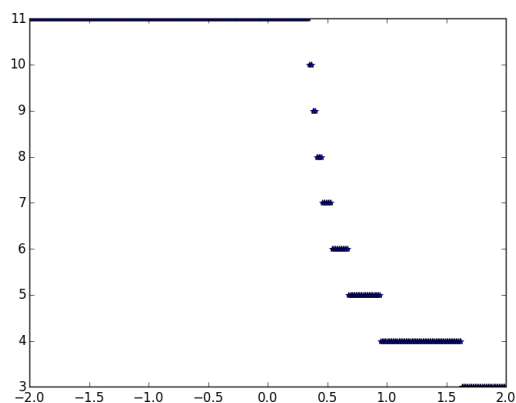
## Exercice 9 – Fractale de Mandelbrot – Corrigé

```
# Question 1
# =====
def suite_u(c,n):
    """
    Calcul de la suite u au rang n.
    Entrées :
    * c(float) : nombre quelconque
    * n(int)
    Sortie :
    * res(float) : valeur de u(n)
    """
    res = 0
    i=0
    while i!=n:
        res = res*res+c
        i=i+1
    return res
```

```
def recherche_k(m,M,c):
    """ Recherche de k """
    k=0
    while k<=m:
        if abs(suite_u(c,k))>M:
            return k
        k=k+1
    return -1
```

```
def fonction_f(m,M,c):
    """ Fonction """
    k = recherche_k(m,M,c)
    if k>=0:
        return k
    else :
        return m+1
```

```
# Question 2
# =====
import matplotlib.pyplot as plt
m,M=10,20
LX = [-2+4*x/400 for x in range(401)]
LF = [fonction_f(m,M,x) for x in LX]
plt.plot(LX,LF,"*")
plt.show()
```



```
# Question 3
# =====
LX = [-2+2.5*x/100 for x in range(101)]
LY = [-1.1+2.2*x/100 for x in range(101)]
XY = [[x,y] for x in LX] for y in LY

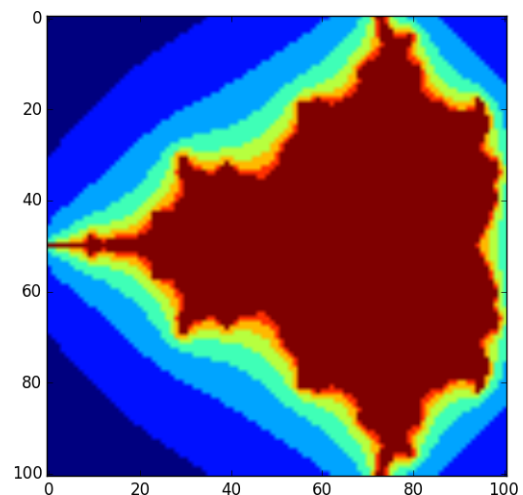
for i in range(len(LX)):
    for j in range(len(LY)):
        XY[i][j]=fonction_f(
            m,M,complex(XY[i][j][0],XY[i][j][1]))
```

```
# Question 4
# =====
res = 100
LX = [-2+2.5*x/res for x in range(res+1)]
LY = [-1.1+2.2*x/res for x in range(res+1)]
XY = [[x,y] for x in LX] for y in LY

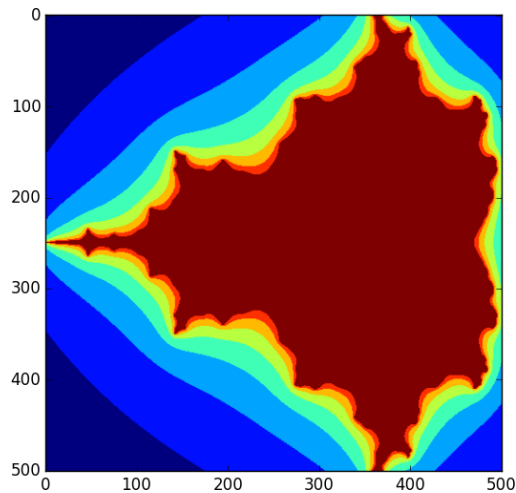
for i in range(len(LX)):
    for j in range(len(LY)):
        XY[i][j]=fonction_f(
            m,M,complex(XY[i][j][0],XY[i][j][1]))

# plt.imshow(XY)
# plt.show()
```

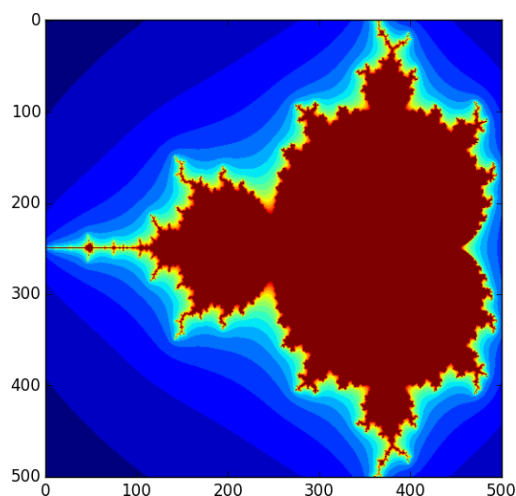
```
# Bilan
# =====
def affichage(m,M,res):
    m = m
    M = M
    LX = [-2+2.5*x/res for x in range(res+1)]
    LY = [-1.1+2.2*x/res for x in range(res+1)]
    XY = [[x,y] for x in LX] for y in LY
    for i in range(len(LX)):
        for j in range(len(LY)):
            XY[i][j]=fonction_f(
                m,M,complex(XY[i][j][0],XY[i][j][1]))
    plt.imshow(XY)
    plt.show()
```



$m = 10, M = 20, 100$  points par 100 points



$m = 10, M = 20, 500 \text{ points par } 500 \text{ points}$



$m = 20, M = 40, 500 \text{ points par } 500 \text{ points}$

## Exercice 10 – Corrigé

```
import numpy as np
# Question 1
R = np.array([[1,2,3],[4,5,6]])
S = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
# Question 2
def test(M):
    """
    Fonction permettant de tester si la
    matrice est carrée et retournant sa taille.
    Entrée :
    * M(numpy.ndarray) : matrice
    Sortie :
    * 0 si taille non carrée
    * n(int) : taille de M si elle est carrée
    """
    l = M.shape[0]
    c = M.shape[1]
    if l==c :
        return l
    else :
        return 0
print(test(R),test(S))
```

```
# Question 3
fid = open("data/ex_006.txt",'r')
M1 = []
for ligne in fid :
    l = ligne.rstrip().split(" ")
    Ligne = [float(x) for x in l]
    M1.append(Ligne)
fid.close()
M1 = np.array(M1)
```

```
# Question 4
if test(M1)>0:
    valeurs_propres = np.linalg.eig(M1)[0]
    print(valeurs_propres)
```

```
# Question 5
def dansIntervalle(L,a,b):
    """
    Vérifier que chaque élément de L est dans
    l'intervalle [a,b]
    Entrées :
    * L(list) : liste de nombres
    * a,b(flt) : nombres
    Sortie :
    * True si chaque élément est dans [a,b]
    * False sinon.
    """
    for e in L :
        if e<a or e> b:
            return False
    return True
print(dansIntervalle(valeurs_propres,0,1))
```

## Exercice 11 – Tri de liste – Corrigé

```
# Question 1
# =====
def comptage(L,n):
    """
    Comptage des éléments de L.
    Entrées :
    * n(int) : entier
    * L(lst) : liste d'éléments inférieurs à n
    """
    P = [0 for i in range(n+1)]
    # P = [0]*(n+1)
    for e in L:
        P[e]=P[e]+1
    return P
from random import randint
maxi = 5
LL = [randint(0,maxi) for x in range(20)]
P = comptage(LL,maxi)
# print(LL)
# print(P)
```

```
# Question 2
# =====
def tri(L,n):
    """
    Tri une liste.
    Entrées :
    * n(int) : entier
    * L(lst) : liste d'éléments inférieurs à n
    Sortie :
    * T(lst) : liste triée.
    """
    P = comptage(L,n)
    T = []
    for i in range(len(P)):
        for j in range(P[i]):
            T.append(i)
    return T
```

```
# Question 3
# =====
from random import randint
maxi = 5
LL = [randint(0,maxi) for x in range(20)]
T = tri(LL,maxi)
print(LL)
print(T)
```

```
# Question 4
# =====
# Complexité quadratique :  $C(n)=O(n+n^2)=O(n^2)$ 
# n : complexité de comptage
#  $n^2$  : complexité des deux boucles imbriquées du
# tri
# Ce tri s'exécutera toujours dans le pire des cas.
# Dans le cas moyen : tri fusion  $O(n \log n)$ 
# Dans le cas moyen : tri insertion  $O(n^2)$ 
```

## Exercice 12 – Corrigé

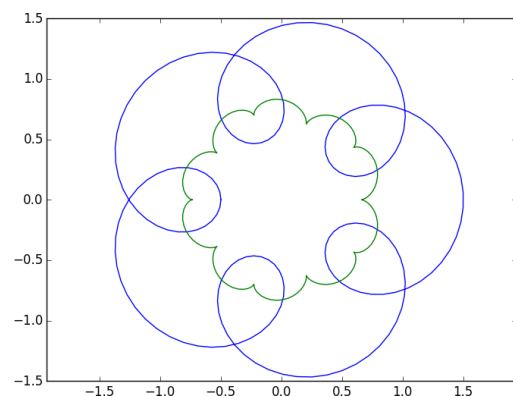
```
b,w = 0.5,6
# Question 1
# =====
import numpy as np
def fonc_p(t):
    return [np.cos(t)+b*np.cos(w*t),np.sin(t)
            +b*np.sin(w*t)]
```

```
def fonc_v(t):
    return [-np.sin(t)-b*w*np.sin(w*t),np.cos(t)
            +b*w*np.cos(w*t)]
```

```
def fonc_a(t):
    return [-np.cos(t)-b*w*w*np.cos(w*t),
            -np.sin(t)-b*w*w*np.sin(w*t)]
```

```
# Question 2
# =====
L=np.linspace(-np.pi,np.pi,200)
```

```
# Question 3
# =====
import matplotlib.pyplot as plt
p = fonc_p(L)
plt.plot(p[0],p[1])
plt.axis("equal")
plt.show()
```



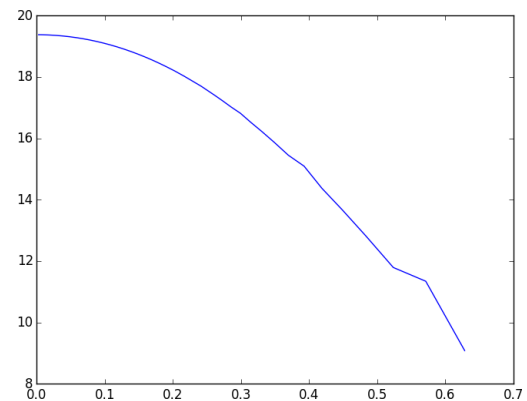
```
# Question 4
# =====
def fonc_d(t):
    xp,yp = fonc_v(t)
    xpp,ypp = fonc_a(t)
    return (xp**2 + yp**2)/(xp*ypp-yp*xpp)
def fonc_c(t):
    fd = fonc_d(t)
    x,y = fonc_p(t)
    xp,yp = fonc_v(t)
    return [x-fd*yp,y+fd*xp]
```

```
# Question 5
# =====
les_xc = []
```

```
les_yc = []
c = fonc_c(L)
plt.plot(c[0],c[1])
```

```
# Question 6
# =====
from math import sqrt
def distance(p):
    """
    Calcule la longueur du profil p.
    Entrée :
    * p(1st) : liste [les_x,les_y]
    Sortie :
    * L(flt) : longueur du profil.
    """
    L=0
    for i in range(len(p[0])-1):
        x0 = p[0][i]
        y0 = p[1][i]
        x1 = p[0][i+1]
        y1 = p[1][i+1]
        L = L+ sqrt((x1-x0)**2+(y1-y0)**2)
    return L
```

```
les_dt = []
les_dist = []
for i in range(10,2000,1) :
    dt = 2*np.pi/i
    L=np.linspace(-np.pi,np.pi,i)
    p = fonc_p(L)
    d = distance(p)
    les_dt.append(dt)
    les_dist.append(d)
plt.plot(les_dt,les_dist)
plt.show()
```



Évolution de la longueur du polynôme en fonction de  $\delta t$ .