Introduction au langage Python

Emmanuel MORAND

juin 2017

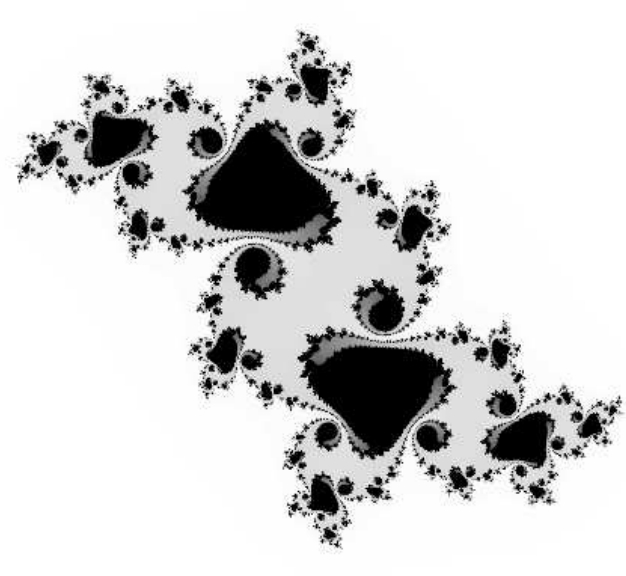

Table des matières

Chapitre 1

Variables et opérations

En Python, les types de variables de base sont les entiers, les flottants, les booléens et les chaînes de caractères. La déclaration et l'initialisation d'une variable Python se réalisent en une seule instruction.

Calcul avec les flottants

Le symbole d'affectation en Python est $=$, les opérateurs $+,-,*$, / et $**$ permettent de calculer somme, différence, produit, quotient et puissance de flottants.

```
>>> x=2
>>> y=8
>>> x/y
0.25
>>> x**y
256
```

Remarque 1. *En Python l'opérateur puissance est $**$, l'opérateur $\hat{}$ est un opérateur bit à bit correspondant au ou exclusif.*

Exercice 1. En utilisant l'interpréteur de commandes Python, déterminer l'entier naturel n et le nombre réel $m \in [1, 2[$ tels que $1696 = m \times 2^n$.

Calcul avec les entiers

Les opérateurs // et % permettent de calculer quotient et reste de la division euclidienne de deux entiers.

```
>>> a,b=25,8
>>> a//b
3
>>> a%b
1
```

Exercice 2. Déterminer le plus grand diviseur commun des entiers 654 et 780 au moyen de l'algorithme d'Euclide.

Calcul avec les booléens

Les opérateurs de comparaison en Python sont == , ! = , < , > , <= et >=.

```
>>>2==3
False
>>>2!=3
True
```

Remarque 2. *En Python, on teste l'égalité de deux variables avec l'opérateur ==, l'opérateur = est l'opérateur d'affectation.*

Exercice 3. Dans quels cas la valeur de l'expression $e\%2 == 1$ est-elle True ou False pour e une variable de type entier ?

Les opérateurs logiques en Python sont **and**, **or** et **not**.

```
>>>not(True) or False
False
```

Exercice 4. Déterminer la table de vérité de NON(A) ET B :

A	B	NON(A) ET B
F	F	...
F	V	...
V	F	...
V	V	...

Opérations sur les chaînes de caractères

Les opérateurs de concaténation sur les chaînes de caractères sont + et ∗.

```
>>> a="bon"
>>> b="jour"
>>> (a+b)*2
'bonjourbonjour'
```

Remarque 3. *En Python, une chaîne de caractères est délimitée par des guillemets.*

La chaîne de caractères vide est représentée par " ", la fonction *len* permet d'obtenir la longueur d'une chaîne de caractères, on accède à l'élément d'indice k d'une chaîne de caractères c par la commande $c[k]$.

```
>>> c="bonjour"
>>> len(c)
7
>>> c[2]
'n'
```

Remarque 4. *En Python, les indices d'une chaîne de caractères c varient entre* 0 *et* $\text{len}(c)-1$.

Chapitre 2

Bibliothèques

Les *bibliothèques* Python regroupent des fonctions utiles que l'on peut utiliser sans avoir à les redéfinir.

Bibliothèque *math*

La bibliothèque ***math*** permet d'utiliser les principales constantes et fonctions mathématiques.

```
>>>from math import*
>>> e,pi,sqrt(2),log(2)
(2.718281828459045, 3.141592653589793, 1.4142135623730951, 0.6931471805599453)
```

Exercice 5. Calculer une valeur approchée de $e^{2\ln 5}$ et de $\cos\left(\frac{\pi}{9}\right)\cos\left(\frac{2\pi}{9}\right)\cos\left(\frac{4\pi}{9}\right)$.

Bibliothèque *fractions*

La bibliothèque ***fractions*** permet le calcul avec des fractions.

```
>>>from fractions import*
>>>Fraction(1,3)-Fraction(1,4)
Fraction(1,12)
```

Exercice 6. Calculer la valeur exacte de $\dfrac{1}{2+\frac{1}{2+\frac{1}{2+\frac{1}{2}}}}$.

Bibliothèque *cmath*

La bibliothèque ***cmath*** permet le calcul avec des nombres complexes.

```
>>>from cmath import*
>>>z=complex(3,4)
>>>z.real,z.imag,abs(z),phase(z)
(3.0,4.0,5.0,0.9272952180016122)
```

Exercice 7. Calculer une valeur approchée de $\mathcal{I}m(e^{i\pi})$.

Certaines fonctions pouvant être définies dans plusieurs bibliothèques, il peut être prudent de préciser la définition que l'on souhaite utiliser.

```
>>>import math,cmath
>>>math.sqrt(4),cmath.sqrt(4)
(2.0,(2+0j))
```

Exercice 8. Comparer les résultats fournis par les bibliothèques math et cmath pour le calcul de $\cos(\pi)$.

Bibliothèque *random*

La bibliothèque ***random*** permet de générer des nombres pseudo-aléatoires.

```
>>>from random import*
>>>random()
0.30346339068560835
>>>randint(1,6)
2
```

Remarque 5. *Avec IDLE, la combinaison de touches Alt+P permet d'obtenir la dernière instruction exécutée.*

Exercice 9. Simuler une suite de 10 lancers au jeu de Pile ou Face avec une pièce équilibrée.

Bibliothèque *sympy*

La bibliothèque ***sympy*** permet le calcul formel.

```
>>> from sympy import*
>>> x=symbols('x')
>>> y=symbols('y')
>>> expand((x+y)**3)
x**3 + 3*x**2*y + 3*x*y**2 + y**3
>>> factor(x**3-y**3)
(x - y)*(x**2 + x*y + y**2)
>>> simplify(x**2/(1+x)-1/(1+x))
x - 1
```

Exercice 10. Factoriser $(ac-bd)^2+(ad+bc)^2$.

Chapitre 3

Programmes

On considère l'algorithme ainsi que le programme Python associé permettant de calculer le maximum de deux nombres réels :

Entrée: variables réelles x et y
Sortie: variable réelle M dont la valeur est égale au maximum des valeurs de x et de y
Début
  Lire x, y
  Si $x < y$ **alors**
    $M \leftarrow y$
  sinon
    $M \leftarrow x$
  FinSi
  Afficher M
Fin

```
#Entrée : x,y flottants
#Sortie : maximum M de x et y
x=float(input("valeur de x?"))
y=float(input("valeur de y?"))
if x<y:
    M=y
else:
    M=x
print("le maximum de x et y est :",M)
```

Remarque 6. *En Python, un bloc d'instructions après un double point commence par une indentation et se termine lors de la désindentation.*

Exercice 11. En utilisant l'environnement de développement IDLE, écrire (*New File*), enregister (*Save*) puis tester le programme précédent (*Run*).

Exercice 12. Écrire un programme permettant de déterminer les solutions réelles d'une équation du second degré puis le tester sur les équations $2x^2 + 3x - 2 = 0$, $4x^2 + 12x + 9 = 0$ et $4x^2 - 4x + 17 = 0$.

Remarque 7. *Avec IDLE, la touches F5 permet d'exécuter le programme.*

Chapitre 4

Boucle for

On considère l'algorithme ainsi que le programme Python associé permettant d'afficher la table de multiplication d'un entier :

Entrée: variable entière n
Sortie: affichage de la table de multiplication de l'entier n
Début
  Lire n
  Pour k **allant de** 1 **à** 10 **faire**
    Afficher n," $*$ ",k," $=$ ",$n*k$
  FinPour
Fin

```
#Entrée : n entier
#Sortie : affichage de la table de multiplication de l'entier n
n=int(input("valeur de n?"))
for k in range(1,11):
    print(n,"*",k,"=",n*k)
```

Remarque 8. *En Python, l'instruction* range(a,b) *permet de générer dans l'ordre croissant les entiers dans l'intervalle* $[a;b[$.

Exercice 13. Créer puis tester un programme permettant d'afficher la liste des diviseurs d'un entier n quelconque.

Exercice 14. Créer puis tester un programme permettant d'obtenir la factorielle $n! = 1 \times 2 \times \cdots \times n$ d'un entier n quelconque.

Exercice 15. Créer puis tester un programme permettant de calculer le n-ième terme de la suite de Fibonacci $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ (chaque terme est la somme des deux termes qui le précèdent) .

Chapitre 5

Boucle while

On considère l'algorithme ainsi que le programme Python associé permettant d'afficher les puissances de deux inférieures à un entier n :

Entrée: variable entière n
Sortie: affichage des puissances de 2 inférieures à n
Début
  Lire n
  $p \leftarrow 1$
  TantQue $p < n$ **faire**
    Afficher p
    $p \leftarrow 2 * p$
  FinTantQue
Fin

```
#Entrée : n entier
#Sortie : affichage des puissances de 2 inférieures à n
n=int(input("valeur de n?"))
p=1
while p<n:
    print(p)
    p=2*p
```

Exercice 16. Créer puis tester un programme permettant de déterminer combien de fois un entier n non nul donné est divisible par 2.

Exercice 17. Créer puis tester un programme permettant de calculer le nombre d'années nécessaire pour doubler un capital placé à intérêts composés avec un taux annuel de $t\%$.

Exercice 18. Créer puis tester un programme permettant de calculer le plus grand diviseur commun de deux entiers m et n au moyen de l'algorithme d'Euclide.

Chapitre 6

Fonctions

On considère l'algorithme de la fonction factorielle ainsi que le programme Python associé :

Fonction: factorielle(n)
Action: Calcul de la factorielle f d'un entier n
Début
  $f \leftarrow 1$
  Pour k **allant de** 1 **à** n **faire**
    $f \leftarrow kf$
  FinPour
  Renvoyer f
Fin

```
def factorielle(n):
    """Calcul de la factorielle f d'un entier n"""
    f=1
    for k in range(1,n+1):
        f=k*f
    return(f)
```

Remarque 9. *En Python, la docstring entre triples guillemets permet de spécifier l'action de la fonction ainsi que ses différents paramètres.*

Exercice 19. Tester le programme précédent en calculant *factorielle(6)* dans l'interpréteur Python.

Exercice 20. Créer puis tester une fonction nommée *pgcd* de paramètres m et n permettant de calculer le plus grand diviseur commun des entiers m et n au moyen de l'algorithme d'Euclide.

Exercice 21. Créer puis tester une fonction *estpremier* de paramètre n permettant de déterminer si un entier $n \geqslant 2$ est un nombre premier.

Exercice 22. En utilisant la fonction précédente, créer puis tester une fonction *premier* de paramètre n permettant de déterminer le n-ième nombre premier.

Chapitre 7

Listes

En python, on représente une liste par la suite de ses éléments séparés par des virgules et encadrée par des crochets, la liste vide est représentée par [ ]. La fonction *len* permet d'obtenir la longueur d'une liste, la fonction *append* permet d'ajouter un nouvel élément en fin de liste, on accède à l'élément d'indice k d'une liste l par la commande $l[k]$.

```
>>> l=[3,2,5,11]
>>> len(l)
4
>>> l[1]
2
>>> l.append(4)
>>> l
[3, 2, 5, 11, 4]
>>>
```

Remarque 10. *En Python, les indices d'une liste l varient entre 0 et $\text{len}(l) - 1$.*

Exercice 23. Créer une fonction *entiers* de paramètre n qui retourne la liste des entiers de 1 à n.

Exercice 24. Créer une fonction *moyenne* de paramètre l qui calcule la moyenne des valeurs d'une liste de nombres l.

Exercice 25. Créer une fonction *maximum* de paramètre l qui calcule le maximum des valeurs d'une liste de nombres l.

Chapitre 8

Tableaux

La bibliothèque ***numpy*** permet le calcul avec des tableaux et la bibliothèque ***matplotlib*** permet de créer des graphiques. Le programme suivant affiche la courbe représentative de la fonction exponentielle sur l'intervalle $[-1;1]$:

```
from numpy import*
from matplotlib.pyplot import*
#nombre de points
N=100
#tableaux de zéros
x=zeros(N)
y=zeros(N)
#création des abscisses et ordonnées
for k in range(0,len(x)):
    x[k]=-1+2*k/(N-1)
    y[k]=exp(x[k])
#création du graphique
plot(x,y)
show()
```

Remarque 11. *Avec numpy, on peut appliquer directement une fonction d'une variable réelle sur un tableau afin d'obtenir le tableau des images.*

```
>>> x=array([1,2,3])
>>> x*(x+1)/2
array([ 1.,  3.,  6.])
```

Exercice 26. On considère une expérience aléatoire dont les résultats possibles sont -1 et 1 avec pour probabilités respectives $\frac{1}{2}$ et $\frac{1}{2}$.

Simuler une série de 1000 expériences et représenter graphiquement le tableau des effectifs cumulés.

Chapitre 9

Codage des caractères

La norme de codage ASCII(American Standard Code for Information Interchange) permet de représenter un caractère informatique par un entier :

```
>>> chr(109)
'm'
>>> ord("n")
110
```

Exercice 27. Créer un programme Python permettant d'afficher la liste des 256 caractères Ascii, en déduire les entiers représentant les 26 lettres de l'alphabet en minuscules, les 26 lettres de l'alphabet en majuscules ainsi que les chiffres de 0 à 9.

Exercice 28. Créer une fonction Python *majuscule* de paramètre c permettant de convertir une chaîne de caractères c de minuscule en majuscule ainsi qu'une fonction Python *minuscule* de paramètre c permettant de convertir une chaîne de caractères c de majuscule en minuscule.

Exercice 29. L'algorithme de chiffrement *ROT13* consiste à décaler chaque lettre d'un message de treize crans dans l'alphabet :

$$\begin{array}{ccc} \text{a} & \mapsto & \text{n} \\ \text{b} & \mapsto & \text{o} \\ & \vdots & \\ \text{y} & \mapsto & \text{l} \\ \text{z} & \mapsto & \text{m} \end{array}$$

Créer une fonction Python *rot13* de paramètre c permettant de réaliser le chiffrement ROT13 d'une chaîne de caractères c.

Chapitre 10

Calcul formel

Étude de fonction

```
>>> from sympy import*
>>> x=symbols('x')
>>> f=exp(x)/(2*x**2+1)
>>> f.subs(x,0)
1
>>> plot(f,(x,-5,5))
<sympy.plotting.plot.Plot object>
>>> limit(f,x,-oo)
0
>>> limit(f,x,+oo)
oo
>>> df=diff(f,x)
>>> solve(df>0,x)
Or(x < -sqrt(2)/2 + 1, x > sqrt(2)/2 + 1)
```

Exercice 30. Étudier les variations de la fonction $f : x \mapsto xe^{-x^2}$.

Développements limités

```
>>> series(exp(x),x,0,5)
1 + x + x**2/2 + x**3/6 + x**4/24 + O(x**5)
```

Exercice 31. Montrer que la courbe représentative de la fonction $f : x \mapsto x\sqrt{\dfrac{x+1}{x-1}}$ admet une asymptote oblique en $+\infty$.

Algèbre linéaire

```
>>> x,y,z=symbols('x'),symbols('y'),symbols('z')
>>> M=Matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> U=Matrix([[x],[y],[z]])
>>> solve(M*U)
{x: z, y: -2*z}
```

Exercice 32. Montrer que $P = \begin{pmatrix} 0 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ est la matrice d'un projecteur dont on déterminera les éléments caractéristiques.

Annexe A

Exercices supplémentaires

Suite de Syracuse

Exercice 33. On considère la suite de Syracuse définie par la relation de récurrence :

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si} \quad u_n \text{ pair} \\ 3u_n + 1 & \text{si} \quad u_n \text{ impair} \end{cases}$$

Représenter graphiquement le *temps de vol* (plus petit indice n tel que $u_n = 1$) en fonction de $u_0 \in [\![1; 100]\!]$.

Méthode de dichotomie

Exercice 34. Déterminer un encadrement de $\sqrt{2}$ d'amplitude 10^{-5} en calculant une valeur approchée du zéro positif de la fonction $x \mapsto x^2 - 2$ au moyen de la méthode de dichotomie.

Méthode des rectangles

Exercice 35. Déterminer un encadrement de π d'amplitude 10^{-5} en encadrant l'intégrale $\displaystyle\int_0^1 \frac{4\,\mathrm{d}x}{1+x^2}$ au moyen de la méthode des rectangles.

Méthode d'Euler

Exercice 36. Déterminer une valeur approchée de e en calculant une valeur approchée de la valeur en 1 de la solution de l'équation différentielle $\begin{cases} y' - y &= 0 \\ y(0) &= 1 \end{cases}$ au moyen de la méthode d'Euler.

Méthode de Monte-Carlo

Exercice 37. Déterminer une valeur approchée de π au moyen de la méthode de Monte-Carlo en simulant le tirage aléatoire d'un point $M(x; y)$ avec $-1 \leqslant x \leqslant 1$ et $-1 \leqslant y \leqslant 1$.

Annexe B

Solutions des exercices

1) On a $1696 = 1,65625 \times 2^{10}$

```
>>> 1696/2**10
1.65625
```

2) Le plus grand diviseur commun de 654 et 780 est 6.

```
>>> 780%654
126
>>> 654%126
24
>>> 126%24
6
>>> 24%6
0
```

3) True pour e impair et False pour e pair.

```
>>> e=7
>>> e%2==1
True
```

4)

A	B	NON(A) ET B
F	F	F
F	V	V
V	F	F
V	V	F

```
>>> A,B=False,False
>>> not(A) and B
False
>>> A,B=False,True
>>> not(A) and B
True
>>> A,B=True,False
>>> not(A) and B
False
>>> A,B=True,True
>>> not(A) and B
False
```

5)

```
>>> exp(2*log(5))
24.999999999999996
>>> cos(pi/9)*cos(2*pi/9)*cos(4*pi/9)
0.12500000000000006
```

6)
```
>>> Fraction(1,2+Fraction(1,2+Fraction(1,2+Fraction(1,2))))
Fraction(12, 29)
```

7)
```
>>> exp(complex(0,1)*pi).imag
1.2246467991473532e-16
```

8)
```
>>> math.cos(math.pi),cmath.cos(cmath.pi)
(-1.0, (-1-0j))
```

9)
```
>>> randint(0,1),randint(0,1),randint(0,1),randint(0,1),randint(0,1),
randint(0,1),randint(0,1),randint(0,1),randint(0,1),randint(0,1)
(0, 1, 0, 0, 1, 1, 1, 1, 1, 1)
```

10) $(ac-bd)^2+(ad+bc)^2=(a^2+b^2)(c^2+d^2)$

```
>>> a,b,c,d=symbols('a'),symbols('b'),symbols('c'),symbols('d')
>>> factor((a*c-b*d)**2+(a*d+b*c)**2)
(a**2 + b**2)*(c**2 + d**2)
```

11)
```
>>>
valeur de x?5
valeur de y?-2.4
le maximum de x et y est : 5.0
```

12)
```
from math import*
#Entrée : a,b,c flottants avec a non nul
#Sortie : solutions de l'équation du second degré ax^2+bx+c=0
a=float(input("valeur de a?"))
b=float(input("valeur de b?"))
c=float(input("valeur de c?"))
d=b**2-4*a*c
if d<0:
    print("l'équation n'admet pas de solution réelle")
else:
    if d==0:
        x0=-b/(2*a)
        print("l'équation admet une unique solution : ",x0)
    else:
        x1=(-b-sqrt(d))/(2*a)
        x2=(-b+sqrt(d))/(2*a)
        print("l'équation admet deux solutions : ",x1,x2)
```

```
>>>
valeur de a?2
valeur de b?3
valeur de c?-2
l'équation admet deux solutions :  -2.0 0.5
>>>
valeur de a?4
valeur de b?12
valeur de c?9
l'équation admet une unique solution :  -1.5
>>>
valeur de a?4
valeur de b?-4
valeur de c?17
l'équation n'admet pas de solution réelle
```

13)
```
#Entrée : n entier
#Sortie : affichage de la liste des diviseurs de l'entier n
n=int(input("valeur de n?"))
print("les diviseurs de",n,"sont")
for k in range(1,n+1):
    if n%k==0:
        print(k)
```

14)
```
#Entrée : n entier
#Sortie : factorielle de l'entier n
n=int(input("valeur de n?"))
f=1
for k in range(2,n+1):
    f=f*k
print("la factorielle de",n,"est",f)
```

15)
```
#Entrée : n entier
#Sortie : calcul du n-ième terme de la suite de Fibonacci
n=int(input("valeur de n?"))
a=1
b=1
for k in range(1,n-1):
    c=a
    a=b
    b=b+c
print("le terme d'ordre",n,"de la suite de Fibonacci est",b)
```

16)
```
#Entrée : n entier
#Sortie : affichage du nombre de fois que l'on peut diviser n par 2
n=int(input("valeur de n?"))
k=n
i=0
while k%2==0:
    k=k//2
    i=i+1
print("le nombre de fois que l'on peut diviser",n,"par",2,"est",i)
```

17)
```
#Entrée : t flottant
#Sortie : affichage du nombre d'années nécessaire pour doubler un capital placé au taux de t%
t=float(input("valeur de t?"))
c=1
i=0
while c<2:
    c=c+c*t/100
    i=i+1
print("le nombre d'années nécessaire pour doubler un capital placé au taux de",t,"%","est",i)
```

18)
```
#Entrée : m,n entiers
#Sortie : plus grand diviseur commun des entiers m et n
m=int(input("valeur de m?"))
n=int(input("valeur de n?"))
if m<n:
    a=m
    b=n
else:
    a=n
    b=m
while(b%a!=0):
    r=b%a
    b=a
    a=r
print("le plus grand diviseur commun de ",m," et ",n," est : ",a)
```

19)
```
>>> factorielle(6)
720
```

20)
```
def pgcd(m,n):
    """Calcul du plus grand diviseur commun des entiers m et n"""
    if m<n:
        a=m
        b=n
    else:
        a=n
        b=m
    while(b%a!=0):
        r=b%a
        b=a
        a=r
    return(a)
```

21)
```
def estpremier(n):
    '''teste si l'entier n est un nombre premier'''
    for k in range(2,n):
        if n%k==0:
            return(False)
    return(True)
```

22)
```
def premier(n):
    '''calcule le n-ième nombre premier'''
    s=0
    p=2
    while s<n:
        if estpremier(p):
            s=s+1
        p=p+1
    return(p-1)
```

23)
```
def entiers(n):
    """retourne la liste des entiers de 1 à n"""
    l=[]
    for k in range(1,n+1):
        l.append(k)
    return(l)
```

24)
```
def moyenne(l):
    '''calcul de la moyenne des valeurs d'une liste de nombres l'''
    s=0
    for k in range(0,len(l)):
        s=s+l[k]
    return(s/len(l))
```

25)
```
def maximum(l):
    '''calcul du maximum des valeurs d'une liste de nombres l'''
    M=l[0]
    for k in range(1,len(l)):
        if l[k]>M:
            M=l[k]
    return(M)
```

26)
```
from numpy import*
from matplotlib.pyplot import*
from random import*
#nombre de points
N=1000
#tableaux de zéros
x=zeros(N+1)
y=zeros(N+1)
#simulation d'une série de N expériences
for k in range(1,N+1):
    x[k]=k
    y[k]=y[k-1]+2*randint(0,1)-1
#création du graphique
plot(x,y)
show()
```

27)
```
for k in range(0,256):
    print(k,chr(k))
```

Alphabet minuscule : de 97 à 122

Alphabet majuscule : de 65 à 90

Chiffres : de 48 à 57

28)
```
def majuscule(c):
    '''conversion de la chaîne de caractères c de minuscule en majuscule'''
    C=""
    for k in range(0,len(c)):
        if 97<=ord(c[k])<=122:
            C=C+chr(ord(c[k])-32)
        else:
            C=C+c[k]
    return(C)
def minuscule(c):
    '''conversion de la chaîne de caractères c de majuscule en minuscule'''
    C=""
    for k in range(0,len(c)):
        if 65<=ord(c[k])<=90:
            C=C+chr(ord(c[k])+32)
        else:
            C=C+c[k]
    return(C)
```

29)
```
def rot13(c):
    '''chiffrement ROT13 de la chaîne de caractères c'''
    C=""
    for k in range(0,len(c)):
        if 97<=ord(c[k])<=122:
            C=C+chr(97+(ord(c[k])-84)%26)
        else:
            C=C+c[k]
    return(C)
```

30)
```
>>> g=x*exp(-x**2)
>>> limit(g,x,-oo)
0
>>> limit(g,x,+oo)
0
>>> dg=diff(g,x)
>>> solve(dg>0,x)
And(-sqrt(2)/2 < re(x), im(x) == 0, re(x) < sqrt(2)/2)
>>> g.subs(x,-sqrt(2)/2)
-sqrt(2)*exp(-1/2)/2
>>> g.subs(x,sqrt(2)/2)
sqrt(2)*exp(-1/2)/2
```

31)
```
>>> x=symbols('x')
>>> f=x*sqrt((x+1)/(x-1))
>>> X=symbols('X')
>>> F=f.subs(x,1/X)
>>> S=series(F,X,0,2)
>>> S.subs(X,1/x)
1/(2*x) + 1 + x + O(x**(-2), (x, oo))
>>> limit(f-(x+1),x,oo)
0
```

32)
```
>>> P=Matrix([[0,-1,-1],[0,1,0],[0,0,1]])
>>> P*P-P
Matrix([
[0, 0, 0],
[0, 0, 0],
[0, 0, 0]])
>>> V=Matrix([[x],[y],[z]])
>>> solve(P*V)
[{z: 0, y: 0}]
>>> solve(P*V-V)
{x: -y - z}
```

33)
```
def syracuse(n):
    '''calcul du temps de vol de la suite de syracuse de premier terme n'''
    u=n
    t=0
    while u!=1:
        if u%2==0:
            u=u/2
        else:
            u=3*u+1
        t=t+1
    return(t)
from numpy import*
from matplotlib.pyplot import*
N=100
x=zeros(N)
y=zeros(N)
for k in range(0,len(x)):
    x[k]=k+1
    y[k]=syracuse(x[k])
plot(x,y,'ro')
show()
```

34)
```
a=1
b=2
while(b-a>10**-5):
    c=(a+b)/2
    if c**2-2>0:
        b=c
    else:
        a=c
print(a,b)
```

35)
```
def encadrement(n):
    '''encadrement de pi au moyen de n rectangles'''
    s1=0
    s2=0
    for k in range(0,n):
        a=k/n
        b=(k+1)/n
        s1=s1+4/(1+b**2)
        s2=s2+4/(1+a**2)
    return([s1/n,s2/n,(s2-s1)/n])
```

```
>>> encadrement(200000)
[3.141587653585585, 3.141597653585585, 1e-05]
```

36)
```
def euler(n):
    '''approximation de e avec n étapes de la méthode d'Euler'''
    y=1
    for k in range(0,n):
        y=y+y/n
    return(y)
```

```
>>> euler(200000)
2.7182750327855842
```

37)
```
from random import*
def montecarlo(n):
    '''approximation de pi avec n simulations d'un tirage aléatoire'''
    s=0
    for k in range(0,n):
        x=2*random()-1
        y=2*random()-1
        if x**2+y**2<1:
            s=s+1
    return(4*s/n)
```

```
>>> montecarlo(200000)
3.14224
```

Annexe C

Image de couverture

```
# -*- coding: utf-8 -*-
import numpy as np
from matplotlib import pyplot as plt
#paramètre de l'ensemble de Julia
c=np.complex(-0.1,0.651)
#nombre d'itérations maximum
Ni=150
#distance d'un point à l'ensemble de Julia
def distance(x,y):
    '''calcule la distance entre 0 et 1 du point de coordonnées (x,y) à l'ensemble de Julia'''
    global Ni
    z=np.complex(x,y)
    i=0
    while np.abs(z)<4 and i<Ni:
        z=z**2+c
        i=i+1
    d=(Ni-i)/Ni
    return(d**0.2)
#taille de l'image
N=1000
image=np.zeros((N,N))
for i in range(0,N):
    for j in range(0,N):
        y=-1.5+3*i/(N-1)
        x=1.5-3*j/(N-1)
        image[i,j]=distance(x,y)
#tracé de l'ensemble de Julia
plt.axis("off")
plt.imshow(image,cmap=plt.cm.gray)
plt.show()
```

Index