

Initiation à l'algorithmique


SOLUTIONS DES TRAVAUX DIRIGÉS

JACQUES TISSEAU

Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr

Avec la participation de ROMAIN BÉNARD, STÉPHANE BONNEAUD, CÉDRIC BUCHE, GIREG DESMEULLES, CÉLINE JOST, SÉBASTIEN KUBICKI, ERIC MAISEL, ALÉXIS NÉDÉLEC, MARC PARENTHOËN et CYRIL SEPTSEAULT.

Ce document regroupe les solutions, testées avec PYTHON 3.2.2, des exercices du cours d'Informatique du 1^{er} semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (ENIB : www.enib.fr). Il accompagne les notes de cours « Initiation à l'algorithmique ».

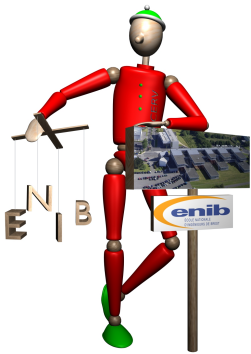


ÉCOLE NATIONALE D'INGÉNIEURS DE BREST

— Cours d'Informatique S1 —

Initiation à l'algorithmique

JACQUES TISSEAU
Ecole nationale d'ingénieurs de Brest
Centre européen de réalité virtuelle
tisseau@enib.fr



Ces notes de cours accompagnent les enseignements d'informatique du 1^{er} semestre (S1) de l'Ecole Nationale d'Ingénieurs de Brest (ENIB : www.enib.fr). Leur lecture ne dispense en aucun cas d'une présence attentive aux cours ni d'une participation active aux travaux dirigés.

Avec la participation de ROMAIN BÉNARD, STÉPHANE BONNEAUD, CÉDRIC BUCHE, GIREG DESMEULLES, CÉLINE JOST, SÉBASTIEN KUBICKI, ERIC MAISEL, ALÉXIS NÉDÉLEC, MARC PARENTHOËN et CYRIL SEPTSEAULT.

version du 21 octobre 2014

Tisseau J., *Initiation à l'algorithmique*, ENIB, cours d'Informatique S1, Brest, 2009.

Sommaire

1	Introduction générale	5
2	Instructions de base	19
3	Procédures et fonctions	39
4	Structures linéaires	67
	Liste des exercices	99

Chapitre 1

Introduction générale

Les exercices de ce chapitre d'introduction ne nécessitent pas de solutions écrites en PYTHON. Cependant, lorsque la solution PYTHON ne fait appel qu'à des instructions aussi simples que celles d'une calculatrice (Td 1.4, 1.5) ou celles de la tortue LOGO (Td 1.1, 1.2, 1.7, 1.10, 1.22, 1.24), celle-ci est proposée afin de se familiariser avec PYTHON.

TD 1.1 : DESSINS SUR LA PLAGE : EXÉCUTION (1)

```
1 # -*- coding: utf-8 -*-
2
3 # avec la tortue Logo
4 from turtle import *
5
6 # avance de 10 pas
7 forward(10)
8
9 # tourne à gauche d'un angle de 120°
10 left(120)
11
12 # avance de 10 pas
13 forward(10)
14
15 # tourne à gauche d'un angle de 120°
16 left(120)
17
18 # avance de 10 pas
19 forward(10)
20
21 # dessin d'un triangle équilatéral de 10 pas de côté
```

TD 1.2 : DESSINS SUR LA PLAGE : CONCEPTION (1)

```
1 # -*- coding: utf-8 -*-
2
3 # avec la tortue Logo
4 from turtle import *
5
6 # dessin du 1er côté
7 forward(10)
8
9 # tourner à gauche de 90°
```

```

10 left(90)
11
12 # dessin du 2ème côté
13 forward(10+2)
14
15 # tourner à gauche de 90°
16 left(90)
17
18 # dessin du 3ème côté
19 forward(10+2+2)
20
21 # tourner à gauche de 90°
22 left(90)
23
24 # dessin du 4ème côté
25 forward(10+2+2+2)
26
27 # tourner à gauche de 90°
28 left(90)
29
30 # dessin du 5ème côté
31 forward(10+2+2+2+2)
32
33 # tourner à gauche de 90°
34 left(90)

```

TD 1.3 : PROPRIÉTÉS D'UN ALGORITHME

```

1 # -*- coding: utf-8 -*-
2
3 print("""
4 validité : si le but était de dessiner un triangle équilatéral
5     de 10 pas de côté, c'est bien ce que l'on obtient.
6
7 robustesse : l'algorithme ne sait que dessiner un triangle
8     équilatéral de 10 pas de côté et ne peut pas être utilisé
9     pour faire autre chose : il est robuste pour dessiner un
10    triangle équilatéral de 10 pas de côté.
11
12 réutilisabilité : l'algorithme ne sait que dessiner un triangle
13    équilatéral de 10 pas de côté et ne peut pas être utilisé
14    pour faire autre chose : il n'est réutilisable tel quel que
15    pour dessiner un triangle équilatéral de 10 pas de côté mais
16    pas pour autre chose (par exemple : dessiner un triangle
17    équilatéral de 20 pas de côté).
18
19 complexité : 30 pas + 2 rotations
20
21 efficacité : on aurait pu choisir de dessiner plus vite
22    (courir au lieu de marcher sur la plage) : instruction speed()
23    pour la tortue Logo.
24    voir : https://docs.python.org/3.2/library/turtle.html#turtle.speed
25 """)

```

TD 1.4 : UNITÉS D'INFORMATION

```

1 # -*- coding: utf-8 -*-
2
3 from math import log, ceil
4
5 # 1 ko (kilooctet)
6 k = 3
7 x = ceil(k*log(10)/log(2))
8 print(k,x,10**k,2**x)
9
10 # 1 Mo (mégaoctet)
11 k = 6
12 x = ceil(k*log(10)/log(2))
13 print(k,x,10**k,2**x)
14
15 # 1 Go (gigaoctet)
16 k = 9
17 x = ceil(k*log(10)/log(2))
18 print(k,x,10**k,2**x)
19
20 # 1 To (téraoctet)
21 k = 12
22 x = ceil(k*log(10)/log(2))
23 print(k,x,10**k,2**x)
24
25 # 1 Po (pétaoctet)
26 k = 15
27 x = ceil(k*log(10)/log(2))
28 print(k,x,10**k,2**x)
29
30 # 1 Eo (exaoctet)
31 k = 18
32 x = ceil(k*log(10)/log(2))
33 print(k,x,10**k,2**x)
34
35 # 1 Zo (zettaoctet)
36 k = 21
37 x = ceil(k*log(10)/log(2))
38 print(k,x,10**k,2**x)
39
40 # 1 Yo (yottaoctet)
41 k = 24
42 x = ceil(k*log(10)/log(2))
43 print(k,x,10**k,2**x)

```

TD 1.5 : PREMIÈRE UTILISATION DE PYTHON

```

1 # -*- coding: utf-8 -*-
2
3 """
4 $ python
5 Python 3.2.2 (default, Nov 21 2011, 16:50:59)
6 [GCC 4.6.2] on linux2
7 Type "help", "copyright", "credits" or "license" for more information.
8 >>> 3+4/5
9 3.8
10 >>> quit
11 Use quit() or Ctrl-D (i.e. EOF) to exit

```

```
12 >>> quit()
13 """
```

TD 1.6 : ERREUR DE SYNTAXE EN PYTHON

```
1 # -*- coding: utf-8 -*-
2
3 print("""
4 >>> x = 3
5 >>> y = 4
6
7 SyntaxError: unexpected indent
8
9 >>> # Il y a un espace de trop devant y = 4
10 """)
```

TD 1.7 : DESSINS SUR LA PLAGE : PERSÉVÉRANCE

```
1 # -*- coding: utf-8 -*-
2
3 # avec la tortue Logo
4 from turtle import *
5
6 # avance de 10 pas
7 forward(10)
8
9 # tourne à gauche d'un angle de 60°
10 left(60)
11
12 # avance de 10 pas
13 forward(10)
14
15 # tourne à gauche d'un angle de 120°
16 left(120)
17
18 # avance de 10 pas
19 forward(10)
20
21 # tourne à gauche d'un angle de 60°
22 left(60)
23
24 # avance de 10 pas
25 forward(10)
26
27 # dessin d'un losange de 10 pas de côté
```

TD 1.8 : AUTONOMIE

```
1 # -*- coding: utf-8 -*-
2
3 """
4 autonomie : http://www.cnrtl.fr/definition/autonomie
5             http://www.cnrtl.fr/antonymie/autonomie
6
7 hétéronomie : http://www.cnrtl.fr/definition/h%C3%A9t%C3%A9ronomie
8 """
```

TD 1.9 : SITE WEB D'INFORMATIQUE S1

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Accueil Mes cours Cycle préparatoire Semestre 1 S1A Algorithmique
5 https://moodle.enib.fr/course/view.php?id=24
6 """

```

TD 1.10 : EXEMPLE DE CONTRÔLE D'ATTENTION (1)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 les 3 principaux thèmes informatiques abordés :
5     instructions, fonctions, séquences
6
7 les 4 principales stratégies pédagogiques suivies :
8     pédagogie par objectifs, par l'exemple, de l'erreur, par problèmes
9
10 les 3 principales qualités comportementales recherchées :
11     rigueur, persévérance, autonomie
12 """

```

TD 1.11 : EXEMPLE DE CONTRÔLE DE TD

```

1 # -*- coding: utf-8 -*-
2
3 # avec la tortue Logo
4 from turtle import *
5
6 # avance de 10 pas
7 forward(10)
8
9 # tourne à gauche d'un angle de 60°
10 left(60)
11
12 # avance de 10 pas
13 forward(10)
14
15 # tourne à gauche d'un angle de 120°
16 left(120)
17
18 # avance de 10 pas
19 forward(10)
20
21 # tourne à gauche d'un angle de 60°
22 left(60)
23
24 # avance de 10 pas
25 forward(10)
26
27 # dessin d'un losange de 10 pas de côté

```

TD 1.12 : EXEMPLE DE CONTRÔLE D'AUTOFORMATION (1)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 a b c : s t
5 0 0 0 : 0 0
6 0 0 1 : 1 0
7 0 1 0 : 1 0
8 0 1 1 : 0 1
9 1 0 0 : 1 0
10 1 0 1 : 0 1
11 1 1 0 : 0 1
12 1 1 1 : 1 1
13 """

```

TD 1.13 : EXEMPLE DE CONTRÔLE DES COMPÉTENCES

```

1 # -*- coding: utf-8 -*-
2
3 """
4 conversions :
5 ga ga = (00)_4
6         = 0
7 bu bu bu = (111)_4
8           = 1*4**2 + 1*4**1 + 1*4**0 = 16 + 4 + 1
9           = 21
10 zo zo zo zo = (2222)_4
11              = 2*4**3 + 2*4**2 + 2*4**1 + 2*4**0
12              = 128 + 32 + 8 + 2
13              = 170
14 meu meu meu meu meu = (33333)_4
15                      = 3*4**4 + 3*4**3 + 3*4**2 + 3*4**1 + 3*4**0
16                      = 768 + 192 + 48 + 12 + 3
17                      = 1023
18
19 opérations :
20 zo zo meu + bu ga meu   = (223)_4 + (103)_4 = (332)_4 = 43 + 19   = 62
21 meu ga meu - bu meu ga  = (303)_4 - (130)_4 = (113)_4 = 51 - 28   = 23
22 zo meu meu * bu ga meu  = (233)_4 * (103)_4 = (31331)_4 = 47 * 19 = 893
23 zo zo zo meu // bu ga zo = (2223)_4 // (102)_4 = (21)_4 = 171 // 18 = 9
24 """

```

TD 1.14 : NOMBRE DE CONTRÔLES

```

1 # -*- coding: utf-8 -*-
2
3 """
4 9 épreuves sur table individuelles de contrôle continu en cours-td
5 (écrit individuel de 15' à 30')
6     CTD : affectation
7     CTD : booléens et tests
8     CTD : codage des nombres
9     CTD : boucles simples

```

```

10     CTD : boucles imbriquées
11     CTD : spécification
12     CTD : appels de fonctions
13     CTD : récursivité
14     CTD : tri
15
16 1 devoir surveillé
17 (écrit individuel de 90' hors grille)
18     DS: instructions de base
19
20 7 épreuves de contrôle continu de laboratoire
21 (sur machine, par binôme)
22 ""

```

TD 1.15 : EXEMPLE DE CONTRÔLE D'AUTOFORMATION (2)

```

1 # -*- coding: utf-8 -*-
2
3 ""
4 (0 < x) and (x < 3) :
5     not ((0 < x) and (x < 3))
6     (not (0 < x)) or (not (x < 3))
7     (x <= 0) or (x >= 3)
8
9 (x < -2) or (x > 4) :
10     not ((x < -2) or (x > 4))
11     (not (x < -2)) and (not (x > 4))
12     (x >= -2) and (x <= 4)
13
14 a and (not b) :
15     not (a and (not b))
16     (not a) or (not (not b))
17     (not a) or b
18
19 (not a) or b :
20     not ((not a) or b)
21     (not (not a)) and (not b)
22     a and (not b)
23 ""

```

TD 1.16 : EXEMPLE DE CONTRÔLE D'ATTENTION (2)

```

1 # -*- coding: utf-8 -*-
2
3 ""
4 les 4 types de contrôle proposés :
5     contrôle d'attention, de laboratoire, d'autoformation, de compétences
6
7 les documents que l'on peut trouver sur le site Web du cours :
8     notes de cours
9     transparents de cours
10    supports de cours
11    questions de cours
12    résultats des évaluations
13 ""

```

TD 1.17 : NOMBRES D'EXERCICES DE TD

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Cet exercice porte le numéro TD 1.17 .
5 C'est donc le 17ème exercice du cours : 16 exercices le précédent.
6 """

```

TD 1.18 : ENVIRONNEMENT DE TRAVAIL

```

1 # -*- coding: utf-8 -*-
2
3 """
4 le type de clavier :
5     AZERTY(UIOP)
6
7 ouverture d'un terminal :
8     $ xterm
9
10 lancement de Python :
11     $ python
12
13 stockage des fichiers de travail :
14     sous le répertoire de connexion de l'utilisateur
15 """

```

TD 1.19 : QCM (1)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Les bonnes réponses sont extraites directement des notes de cours :
5 1a, 2e, 3b, 4b, 5e, 6e, 7a, 8d
6 """

```

TD 1.20 : PUISSANCE DE CALCUL

```

1 # -*- coding: utf-8 -*-
2
3 """
4 L'unité de puissance est donné ici en Mips
5 (million d'instructions par seconde).
6
7     le premier micro-ordinateur de type PC : 10(-1) Mips
8
9     une console de jeu actuelle : 104 Mips
10
11     un micro-ordinateur actuel : 104 Mips
12
13     Deeper-Blue : l'ordinateur qui a « battu » Kasparov aux
14     échecs en 1997 : 10(7) Mips
15
16     le plus puissant ordinateur actuel : 109 Mips
17 """

```

TD 1.21 : STOCKAGE DE DONNÉES

```

1 # -*- coding: utf-8 -*-
2
3 """
4 une page d'un livre : 50 lignes de 80 caractères = 4 ko
5
6 une encyclopédie en 20 volumes : 20 volumes de 1000
7     pages de 50 lignes de 80 caractères = 80 Mo (sans images !)
8
9 une photo couleur : de 1Mo à 100 Mo selon la qualité
10
11 une heure de vidéo : de 500 Mo à 2 Go selon la qualité vidéo (DivX, MPEG-2...)
12
13 une minute de son : de 1 Mo (MP3) à 10 Mo selon la qualité du son
14
15 une heure de son : de 60 Mo à 600 Mo selon la qualité du son
16 """

```

TD 1.22 : DESSINS SUR LA PLAGE : EXÉCUTION (2)

```

1 # -*- coding: utf-8 -*-
2
3 # avec la tortue Logo
4 from turtle import *
5
6 # avance de 3 pas
7 forward(3)
8
9 # tourne à gauche d'un angle de 90°
10 left(90)
11
12 # avance de 4 pas
13 forward(4)
14
15 # rejoindre le point de départ
16 goto(0,0)
17
18 """
19 On trace un triangle rectangle dont l'hypothénuse fait 5 pas de long
20     (d'après le théorème de Pythagore :  $5^2 = 3^2 + 4^2$ ).
21 On a donc marché  $3 + 4 + 5 = 12$  pas.
22 """

```

TD 1.23 : DESSINS SUR LA PLAGE : CONCEPTION (2)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Imaginons l'algorithme de tracé suivant :
5
6     avance de 2 pas,
7     tourne à gauche de 90°,
8     avance de 3 pas,

```

```

9     tourne à gauche de 90°,
10    avance de 4 pas,
11    tourne à gauche de 90°,
12    avance de 5 pas,
13    tourne à gauche de 90°,
14    avance de 6 pas.
15
16 validité : on doit au moins vérifier que la figure obtenue à toutes
17    les caractéristiques recherchées : spirale rectangulaire de
18    5 côtés, le plus petit côté faisant 2 pas de long et chaque
19    côté fait un pas de plus que le précédent.
20
21 robustesse : cet algorithme suppose qu'on a suffisamment de place pour
22    tracer une spirale (le dernier côté fait 6 pas de long);
23    s'il fonctionne correctement sur une plage,
24    il ne fonctionnera certainement plus dans un placard.
25
26 réutilisabilité : il existe une infinité de spirales rectangulaires
27    qui ont les caractéristiques attendues; il suffit de penser
28    à changer l'orientation initiale ou la longueur du pas par exemple.
29    On ne pourra donc pas utiliser l'algorithme tel quel dans toutes les
30    configurations : il aurait fallu paramétrer l'angle de rotation et la
31    longueur du pas.
32
33 complexité : on peut la caractériser par le nombre de pas :
34    2 + 3 + 4 + 5 + 6 = 20 pas.
35
36 efficacité : si la complexité se calcule en nombre de pas comme
37    ci-dessus, on pourrait imaginer par exemple que la fréquence des
38    pas soit plus grande (« fréquence d'horloge ») ou que
39    5 personnes prennent en charge chacune un côté de la spirale
40    pour gagner du temps (« système multi-processeurs »).
41 """

```

TD 1.24 : TRACÉS DE POLYGONES RÉGULIERS

```

1 # -*- coding: utf-8 -*-
2
3 # avec la tortue Logo
4 from turtle import *
5
6 # Pentagone régulier de 10 pas de côté
7 # avance de 10 pas
8 forward(10)
9 # tourne à gauche de (360/5)°
10 left(360/5)
11 # avance de 10 pas
12 forward(10)
13 # tourne à gauche de (360/5)°
14 left(360/5)
15 # avance de 10 pas
16 forward(10)
17 # tourne à gauche de (360/5)°
18 left(360/5)
19 # avance de 10 pas
20 forward(10)
21 # tourne à gauche de (360/5)°
22 left(360/5)

```

```

23 # avance de 10 pas
24 forward(10)
25 # tourne à gauche de (360/5)°
26 left(360/5)
27
28 # Pentagone régulier de 10 pas de côté
29 # répète 5 fois de suite
30 for i in [1,2,3,4,5] :
31 # les 2 instructions
32 # avance de 10 pas,
33     forward(10)
34 # tourne à gauche de (360/5)°
35     left(360/5)
36
37 # Hexagone régulier de 10 pas de côté
38 # répète 6 fois de suite
39 for i in [1,2,3,4,5,6] :
40 # les 2 instructions
41 # avance de 10 pas,
42     forward(10)
43 # tourne à gauche de (360/6)°
44     left(360/6)
45
46 # Octogone régulier de 10 pas de côté
47 # répète 8 fois de suite
48 for i in [1,2,3,4,5,6,7,8] :
49 # les 2 instructions
50 # avance de 10 pas,
51     forward(10)
52 # tourne à gauche de (360/8)°
53     left(360/8)
54
55 # Polygone régulier de n côtés de 10 pas chacun
56 # répète n fois de suite (par exemple n = 113)
57 n = 113
58 for i in range(n) :
59 # les 2 instructions
60 # avance de 10 pas,
61     forward(10)
62 # tourne à gauche de (360/n)°
63     left(360/n)

```

TD 1.25 : LA MULTIPLICATION « À LA RUSSE »

```

1 # -*- coding: utf-8 -*-
2
3 """
4 64*96 = 6144
5     M   m   r   p = (M*r + p)
6     96   64  0  0
7     192  32  0  0
8     384  16  0  0
9     768   8  0  0
10    1536  4  0  0
11    3072  2  0  0
12    6144  1  1 6144
13
14

```

```

15 45*239 = 10755
16   M   m   r   p = (M*r + p)
17   239 45 1 239
18   478 22 0 239
19   956 11 1 1195
20   1912 5 1 3107
21   3824 2 0 3107
22   7648 1 1 10755
23 ""

```

TD 1.26 : LA MULTIPLICATION ARABE

```

1 # -*- coding: utf-8 -*-
2
3 ""
4 63247*124 = 7842628
5
6   7   4   2   3   6
7   8/2 6/1 8/0 2/1 4/2 : 4
8   8 4/1 8/0 4/0 6/0 2/1 : 2
9   2 7/0 4/0 2/0 3/0 6/0 : 1
10  6
11  2   4   8   7   0
12 ""

```

TD 1.27 : LA DIVISION CHINOISE

```

1 # -*- coding: utf-8 -*-
2
3 ""
4 1234 = 176*7 + 2
5
6 1 0000
7 2 1234
8
9 règle 7-1 :
10   1 0100
11   2 1034
12
13 règle 7-5 :
14   1 0100
15   2 1244
16
17 règle 7-4 :
18   1 0111
19   2 1204
20
21 règle 7-7 :
22   1 0110
23   2 1212
24 ""

```

TD 1.28 : LE CALCUL SHADOK

```

1 # -*- coding: utf-8 -*-
2
3 """
4 conversions :
5 ga ga = (00)_4
6         = 0
7 bu bu bu = (111)_4
8           = 1*4**2 + 1*4**1 + 1*4**0 = 16 + 4 + 1
9           = 21
10 zo zo zo zo = (2222)_4
11              = 2*4**3 + 2*4**2 + 2*4**1 + 2*4**0
12              = 128 + 32 + 8 + 2
13              = 170
14 meu meu meu meu meu = (33333)_4
15                       = 3*4**4 + 3*4**3 + 3*4**2 + 3*4**1 + 3*4**0
16                       = 768 + 192 + 48 + 12 + 3
17                       = 1023
18
19 opérations :
20 zo zo meu + bu ga meu    = (223)_4 + (103)_4 = (332)_4 = 43 + 19    = 62
21 meu ga meu - bu meu ga   = (303)_4 - (130)_4 = (113)_4 = 51 - 28    = 23
22 zo meu meu * bu ga meu   = (233)_4 * (103)_4 = (31331)_4 = 47 * 19   = 893
23 zo zo zo meu // bu ga zo = (2223)_4 // (102)_4 = (21)_4 = 171 // 18 = 9
24 """

```

Chapitre 2

Instructions de base

TD 2.1 : UNITÉ DE PRESSION

```
1 # -*- coding: utf-8 -*-
2
3 nTorr = 1
4 nPa = nTorr*101325/760
5 print(nTorr,'->',nPa)
```

TD 2.2 : SUITE ARITHMÉTIQUE (1)

```
1 # -*- coding: utf-8 -*-
2
3 a, r, n = 0, 1, 5
4 s = a*(n+1) + r*n*(n+1)//2
5 print('somme arithmétique :',a,r,n,'->',s)
```

TD 2.3 : PERMUTATION CIRCULAIRE (1)

```
1 # -*- coding: utf-8 -*-
2
3 x, y, z, t = 1, 2, 3, 4
4 print('avant :',x,y,z,t)
5
6 x, y, z, t = t, x, y, z
7 print('après :',x,y,z,t)
```

TD 2.4 : SÉQUENCE D'AFFECTIONS (1)

```
1 # -*- coding: utf-8 -*-
2
3 # division entière : a = b*q + r
4 a = 19
5 b = 6
6 q = 0
7 r = a
8 print('avant :',a,b,q,r)
9
10 r = r - b
11 q = q + 1
```

```

12 r = r - b
13 q = q + 1
14 r = r - b
15 q = q + 1
16 print('après :',a,b,q,r)

```

TD 2.5 : OPÉRATEURS BOOLÉENS DÉRIVÉS (1)

```

1 # -*- coding: utf-8 -*-
2
3 # opérateur booléen 1 : ou exclusif
4 a, b = 1, 0
5 y = (a and not b) or (not a and b)
6 print('ou exclusif :',a,b,'->',int(y))
7
8 # opérateur booléen 2 : non ou
9 a, b = 1, 0
10 y = not (a or b)
11 print('non ou      :',a,b,'->',int(y))
12
13 # opérateur booléen 3 : non et
14 a, b = 1, 0
15 y = not (a and b)
16 print('non et     :',a,b,'->',int(y))
17
18 # opérateur booléen 4 : implication
19 a, b = 1, 0
20 y = not a or b
21 print('implication :',a,b,'->',int(y))
22
23 # opérateur booléen 5 : équivalence
24 a, b = 1, 0
25 y = (not a or b) and (not b or a)
26 print('équivalence :',a,b,'->',int(y))

```

TD 2.6 : CIRCUIT LOGIQUE (1)

```

1 # -*- coding: utf-8 -*-
2
3 a, b, c = 0, 1, 0
4 u = a and c
5 v = b and not c
6 s = u or v
7 print(a,b,c,'->',int(s))

```

TD 2.7 : LOIS DE DE MORGAN

```

1 # -*- coding: utf-8 -*-
2
3 # loi de De Morgan 1
4 a, b = 0, 0
5 y = (not (a or b)) == (not a and not b)
6 print(a,b,'->',int(y))
7 a, b = 0, 1

```

```

8 y = (not (a or b)) == (not a and not b)
9 print(a,b,'->',int(y))
10 a, b = 1, 0
11 y = (not (a or b)) == (not a and not b)
12 print(a,b,'->',int(y))
13 a, b = 1, 1
14 y = (not (a or b)) == (not a and not b)
15 print(a,b,'->',int(y))
16 print()
17
18 # loi de De Morgan 2
19 a, b = 0, 0
20 y = (not (a and b)) == (not a or not b)
21 print(a,b,'->',int(y))
22 a, b = 0, 1
23 y = (not (a and b)) == (not a or not b)
24 print(a,b,'->',int(y))
25 a, b = 1, 0
26 y = (not (a and b)) == (not a or not b)
27 print(a,b,'->',int(y))
28 a, b = 1, 1
29 y = (not (a and b)) == (not a or not b)
30 print(a,b,'->',int(y))

```

TD 2.8 : MAXIMUM DE 2 NOMBRES

```

1 # -*- coding: utf-8 -*-
2
3 x, y = 10, 13
4 if x > y: m = x
5 else    : m = y
6 print(x,y,'->',m)

```

TD 2.9 : FONCTION « PORTE »

```

1 # -*- coding: utf-8 -*-
2
3 x = -0.25
4 if x < -1 or x > 2 : y = 0
5 else               : y = 2
6 print(x,'->',y)

```

TD 2.10 : OUVERTURE D'UN GUICHET

```

1 # -*- coding: utf-8 -*-
2
3 jour, h = 'mardi', 13.5
4 if h < 8 or (13 < h < 14) or h > 17 : guichet = 'fermé'
5 else:
6     if jour == 'dimanche' : guichet = 'fermé'
7     else:
8         if jour == 'samedi' and (h > 13 or h < 8) : guichet = 'fermé'
9         else: guichet = 'ouvert'
10 print(jour,h,'->',guichet)

```

TD 2.11 : CATÉGORIE SPORTIVE

```
1 # -*- coding: utf-8 -*-
2
3 age = 9.5
4 if 6 <= age < 8 : categorie = 'poussin'
5 elif 8 <= age < 10 : categorie = 'pupille'
6 elif 10 <= age < 12 : categorie = 'minime'
7 elif 12 <= age < 14 : categorie = 'cadet'
8 else : categorie = 'autre'
9 print(age,'->',categorie)
```

TD 2.12 : DESSIN D'ÉTOILES (1)

```
1 # -*- coding: utf-8 -*-
2
3 n = 6
4
5 i = n
6 while i > 0:
7     print(i*'*')
8     i = i - 1
```

TD 2.13 : FONCTION FACTORIELLE

```
1 # -*- coding: utf-8 -*-
2
3 n = 7
4
5 i = 1
6 f = 1
7 while i <= n:
8     f = f*i
9     i = i + 1
10 print(n,'->',f)
```

TD 2.14 : FONCTION SINUS

```
1 # -*- coding: utf-8 -*-
2
3 from math import *
4
5 x = pi/2
6
7 k = 0
8 terme = x
9 somme = terme
10 s = 1.e-9
11 while fabs(terme) > s:
12     terme = -terme*x*x/((2*k+2)*(2*k+3))
13     somme = somme + terme
14     k = k + 1
15 print('sin :',x,somme,':',sin(x))
```

TD 2.15 : ALGORITHME D'EUCLIDE

```

1 # -*- coding: utf-8 -*-
2
3 a, b = 15, 21
4
5 L, l = a, b
6 r = L%l
7 while r > 0 :
8     L = l
9     l = r
10    r = L%l
11 print(a,b,'->',l)

```

TD 2.16 : DIVISION ENTIÈRE

```

1 # -*- coding: utf-8 -*-
2
3 a, b = 18, 7
4
5 q, r = 0, a
6 while r > b :
7     r = r - b
8     q = q + 1
9 print(a,b,'->',q,r)

```

TD 2.17 : AFFICHAGE INVERSE

```

1 # -*- coding: utf-8 -*-
2
3 s = "chaîne"
4
5 i = len(s) - 1
6 while i >= 0:
7     print(s[i])
8     i = i - 1

```

TD 2.18 : PARCOURS INVERSE

```

1 # -*- coding: utf-8 -*-
2
3 s = (0,1,2,3,4,5,6,7,8)
4
5 i = len(s) - 1
6 while i >= 0 :
7     print(s[i],end='')
8     i = i - 1

```

TD 2.19 : SUITE ARITHMÉTIQUE (2)

```
1 # -*- coding: utf-8 -*-
2
3 # suite arithmétique 1 : version itérative
4 # n additions
5 n = 12573
6
7 i, somme = 0, 0
8 while i <= n :
9     somme = somme + i
10    i = i + 1
11 print('itérative : ',n,'->',somme)
12
13 # suite arithmétique 2 : version constante
14 # 3 opérations
15 n = 12573
16
17 somme = n*(n+1)//2
18 print('constante : ',n,'->',somme)
```

TD 2.20 : DESSIN D'ÉTOILES (2)

```
1 # -*- coding: utf-8 -*-
2
3 n = 6
4
5 i = n
6 while i > 0:
7     j = i
8     while j > 0:
9         print('*',end='')
10        j = j - 1
11    print()
12    i = i - 1
```

TD 2.21 : OPÉRATEURS BOOLÉENS DÉRIVÉS (2)

```
1 # -*- coding: utf-8 -*-
2
3 # opérateur booléen 1 : ou exclusif
4 for a in [0,1]:
5     for b in [0,1]:
6         y = (a and not b) or (not a and b)
7         print('ou exclusif : ',a,b,'->',int(y))
8 print()
9
10 # opérateur booléen 2 : non ou
11 for a in [0,1]:
12     for b in [0,1]:
13         y = not (a or b)
14         print('non ou      : ',a,b,'->',int(y))
15 print()
16
17 # opérateur booléen 3 : non et
18 for a in [0,1]:
19     for b in [0,1]:
```



```

20     y = not (a and b)
21     print('non et      : ',a,b,'->',int(y))
22 print()
23
24 # opérateur booléen 4 : implication
25 for a in [0,1]:
26     for b in [0,1]:
27         y = not a or b
28         print('implication : ',a,b,'->',int(y))
29 print()
30
31 # opérateur booléen 5 : équivalence
32 for a in [0,1]:
33     for b in [0,1]:
34         y = (not a or b) and (not b or a)
35         print('équivalence : ',a,b,'->',int(y))
36 print()

```

TD 2.22 : DAMIER

```

1 # -*- coding: utf-8 -*-
2
3 from turtle import *
4
5 n, m = 5, 3
6 dx, dy = 30, 20
7 x0, y0 = 0, 0
8 for i in range(n+1):
9     up()
10    goto(x0+i*dx,y0)
11    setheading(90)
12    down()
13    forward(m*dy)
14 for j in range(m+1):
15     up()
16     goto(x0,y0+j*dy)
17     setheading(0)
18     down()
19     forward(n*dx)

```

TD 2.23 : TRACE DE LA FONCTION FACTORIELLE

```

1 # -*- coding: utf-8 -*-
2
3 n = 7
4
5 i = 1
6 f = 1
7 print(n,i,f)
8 while i <= n:
9     f = f*i
10    i = i + 1
11    print(n,i,f)
12 print(n,i,f)
13
14 print(n,'->',f)

```

TD 2.24 : FIGURE GÉOMÉTRIQUE

```

1 # -*- coding: utf-8 -*-
2
3 from turtle import *
4 x0 = 0
5 y0 = 0
6 r = 10
7 n = 5
8 m = 10
9 for i in range(n) :
10     up()
11     y = y0 - 2*r*i
12     x = x0 + r*(i%2)
13     goto(x,y)
14     for j in range(m) :
15         down()
16         circle(r)
17         up()
18         x = x + 2*r
19         goto(x,y)

```

TD 2.25 : SUITE ARITHMÉTIQUE (3)

```

1 # -*- coding: utf-8 -*-
2
3 # suite arithmétique 1 : version itérative
4 n = 10
5
6 #initialisation
7 i, s = 0, 0
8 # invariant : s = somme des i premiers entiers
9 print(n,i,s)
10 # condition d'arrêt : i >= n
11 while not i >= n :
12     i = i + 1
13     s = s + i
14     # invariant : s = somme des i premiers entiers
15     print(n,i,s)
16 print('itérative : ',n,'->',s)

```

TD 2.26 : QCM (2)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Les bonnes réponses sont extraites directement des notes de cours :
5 1c, 2b, 3c, 4a, 5c, 6a, 7d, 8a, 9d, 10d, 11c, 12a, 13b, 14a, 15b
6 """

```

TD 2.27 : UNITÉ DE LONGUEUR

```

1 # -*- coding: utf-8 -*-
2
3 # AL : année-lumière
4 # M  : mètre
5
6 dAL = 1
7 dM = dAL/(365.25*3600*24*299792458)
8 print(dM)
9
10 dM = 1
11 dAL = dM*(365.25*3600*24*299792458)
12 print(dAL)

```

TD 2.28 : PERMUTATION CIRCULAIRE (2)

```

1 # -*- coding: utf-8 -*-
2
3 x, y, z = 1, 2, 3
4 print('avant :',x,y,z)
5
6 # permutation circulaire gauche
7 x, y, z = y, z, x
8
9 print('après :',x,y,z)

```

TD 2.29 : SÉQUENCE D'AFFECTIONS (2)

```

1 # -*- coding: utf-8 -*-
2
3 n = 1
4 s = n
5 print('avant :',n,s)
6
7 n = n + 1
8 s = s + n
9 n = n + 1
10 s = s + n
11 n = n + 1
12 s = s + n
13 n = n + 1
14 s = s + n
15
16 print('après :',n,s)

```

TD 2.30 : CIRCUITS LOGIQUES (2)

```

1 # -*- coding: utf-8 -*-
2
3 a, b, c = 1, 0, 1
4
5 # circuit 1
6 s = (a and not b) or (b and not a)
7 print('circuit 1 :',a,b,'->',int(s))
8

```

```

9 # circuit 2
10 s = not (not (a and not b) and not (b and not a))
11 print('circuit 2 :',a,b,'->',int(s))
12
13 # circuit 3
14 s = (a and not b) or (not a and b)
15 t = (not a) and b
16 print('circuit 3 :',a,b,'->',int(s),int(t))
17
18 # circuit 4
19 s = (a != (b != c))
20 t = (b and c) or (a and (b != c))
21 print('circuit 4 :',a,b,'->',int(s),int(t))
22
23 # circuit 5
24 s = (a and c) or (b and c) or (a and b)
25 print('circuit 5 :',a,b,c,'->',int(s))
26
27 # circuit 6
28 u = a and (not b) and c
29 s = (a and not c) or u
30 t = (b and c) or u
31 print('circuit 6 :',a,b,c,'->',int(s),int(t))
32
33 # circuit 7
34 u = not a and not b and not c
35 v = a and not b and not c
36 w = a and b and not c
37 s = not (not u and not v and not w)
38 print('circuit 7 :',a,b,c,'->',int(s))
39
40 # circuit 8
41 s7 = a and b and c
42 s6 = a and b and not c
43 s5 = a and not b and c
44 s4 = a and not b and not c
45 s3 = not a and b and c
46 s2 = not a and b and not c
47 s1 = not a and not b and c
48 s0 = not a and not b and not c
49 print('circuit 8 :',a,b,c,'->',int(s0),int(s1),int(s2),int(s3),int(s4),int(s5),int(s6),int(s7))

```

TD 2.31 : ALTERNATIVE SIMPLE ET TEST SIMPLE

```

1 # -*- coding: utf-8 -*-
2
3 x = -1
4 if x < 0: x = 3
5 else: x = -x
6 print('alternative simple :',x)
7
8 x = -1
9 if x < 0: x = 3
10 if not x < 0: x = -x
11 print('2 tests simples :',x)

```

TD 2.32 : RACINES DU TRINOME

```

1 # -*- coding: utf-8 -*-
2
3 from math import *
4
5 a, b, c = 1, 0, -1
6
7 delta = b*b - 4*a*c
8 if delta > 0 :
9     racines = [(-b - sqrt(delta))/(2*a), (-b + sqrt(delta))/(2*a)]
10 elif delta == 0 :
11     racines = [-b/(2*a)]
12 else :
13     racines = []
14
15 print(a,b,c,':',racines)

```

TD 2.33 : SÉQUENCES DE TESTS

```

1 # -*- coding: utf-8 -*-
2
3 # séquence 1
4 x = -3
5 if x < 0 : x = -x
6 print('sequence 1 : ',x)
7
8 # séquence 2
9 x0 = 3
10 x = 5
11 if x < x0 : y = -1
12 else : y = 1
13 print('sequence 2 : ',y)
14
15 # séquence 3
16 p = 1
17 d = 0
18 r = 0
19 h = 1
20 z = 0
21 f = p and (d or r)
22 g = not r
23 m = not p and not z
24 g = g and (d or h or m)
25 if f or g : y = 1
26 else : y = 0
27 print('sequence 3 : ',y)
28
29 # séquence 4
30 x = 2
31 y = 3
32 d = 5
33 h = 4
34 if x > 0 and x < d :
35     if y > 0 and y < h : ok = 1
36     else : ok = 0
37 else : ok = 0
38 print('sequence 4 : ',ok)
39

```

```
40 # séquence 5
41 x = 3
42 y = -2
43 if x < y : y = y - x
44 elif x == y : y = 0
45 else : y = x - y
46 print('séquence 5 :',y)
```

TD 2.34 : RACINE CARRÉE ENTIÈRE

```
1 # -*- coding: utf-8 -*-
2
3 n = 8
4
5 r = 0
6 while (r+1)**2 <= n :
7     r = r + 1
8
9 print(n,r)
```

TD 2.35 : EXÉCUTIONS D'INSTRUCTIONS ITÉRATIVES

```
1 # -*- coding: utf-8 -*-
2
3 # itérations 1
4 print('--- itérations 1:')
5 x = 0
6 while x != 33:
7     x = int(input('entrer un nombre : '))
8
9 # itérations 2
10 print('--- itérations 2:')
11 x = 0
12 while x <= 0 or x > 5 :
13     x = int(input('entrer un nombre : '))
14
15 # itérations 3
16 print('--- itérations 3: somme de 5 nombres')
17 s = 0
18 for i in range(5) :
19     x = int(input('entrer un nombre : '))
20     s = s + x
21 print(s)
22
23 # itérations 4
24 print('--- itérations 4: triangle d\'étoiles')
25 for i in range(0,10) :
26     for j in range(0,i) :
27         print('*',end=' ')
28     print()
29
30 # itérations 5
31 print('--- itérations 5: triangle d\'étoiles inversé')
32 for i in range(0,10) :
33     j = 10 - i
34     while j > 0 :
```

```

35         print('*',end=' ')
36         j = j - 1
37     print()
38
39 # itérations 6
40 print('--- itérations 6: tables de multiplication')
41 for i in range(1,10):
42     for j in range(0,11) :
43         print(i, 'x', j, ' = ', i*j)
44     print()
45
46 # itérations 7
47 print('--- itérations 7: triangle de Pascal')
48 for n in range(10) :
49     for p in range(n+1) :
50         num = 1
51         den = 1
52         for i in range(1,p+1) :
53             num = num*(n-i+1)
54             den = den*i
55         c = num//den
56         print(c,end=' ')
57     print()
58
59 # itérations 8
60 print('--- itérations 8: nombres de Fibonacci')
61 for n in range(0,15) :
62     f, f1, f2 = 1, 1, 1
63     for i in range(2,n+1) :
64         f = f1 + f2
65         f2 = f1
66         f1 = f
67     print(f,end=' ')
68 print()
69
70 # itérations 9
71 print('--- itérations 9: codage binaire')
72 b = 2
73 k = 8
74 n = 23
75 s = 0
76 i = k - 1
77 q = n
78 while q != 0 and i >= 0 :
79     s = s + (q%b)*b**(k-1-i)
80     print(q%b,end=' ')
81     q = q//b
82     i = i - 1

```

TD 2.36 : FIGURES GÉOMÉTRIQUES

```

1 # -*- coding: utf-8 -*-
2
3 from math import pi
4
5 # rectangle
6 l, L = 2, 3
7 p = 2*(L+1)

```

```

8 s = L*l
9 print('rectangle :',l,L,'->',p,s)
10
11 # cercle
12 r = 1
13 p = 2*pi*r
14 s = pi*r*r
15 print('cercle      :',r,'->',p,s)
16
17 # cylindre
18 r, h = 1, 2
19 s = 2*pi*r*h
20 v = pi*r*r*h
21 print('cylindre   :',r,h,'->',s,v)
22
23 # sphère
24 r = 1
25 s = 4*pi*r*r
26 v = 4*pi*r*r*r/3
27 print('sphère     :',r,'->',s,v)

```

TD 2.37 : SUITES NUMÉRIQUES

```

1 # -*- coding: utf-8 -*-
2
3 # somme arithmétique : version constante
4 a, b, n = 0, 1, 5
5 s = a*(n+1) + b*n*(n+1)//2
6 print('somme arithmétique :',a,b,n,'->',s)
7
8 # somme arithmétique : version itérative
9 a, b, n = 0, 1, 5
10 s = 0
11 for i in range(n+1):
12     s = s + a + b*i
13 print('somme arithmétique :',a,b,n,'->',s)
14
15 # somme géométrique : version constante
16 a, b, n = 1, 2, 5
17 s = a*(b**(n+1) - 1)/(b-1)
18 print('somme géométrique :',a,b,n,'->',s)
19
20 # somme géométrique : version itérative
21 a, b, n = 1, 2, 5
22 s = 0
23 for i in range(n+1):
24     s = s + a*b**i
25 print('somme géométrique :',a,b,n,'->',s)

```

TD 2.38 : CALCUL VECTORIEL

```

1 # -*- coding: utf-8 -*-
2
3 from math import *
4
5 # cosinus directeurs

```

```

6 x, y, z = 1, 1, sqrt(2)
7 r = sqrt(x*x + y*y + z*z)
8 a1 = x/r
9 a2 = y/r
10 a3 = z/r
11 print('cosinus directeurs :',x,y,z,'->',a1,a2,a3)
12
13 # produit scalaire
14 x1, y1, z1 = 1, 1, 1
15 x2, y2, z2 = -1, -1, 2
16 p = x1*x2 + y1*y2 + z1*z2
17 print('produit scalaire :',x1,y1,z1,x2,y2,z2,'->',p)
18
19 # produit vectoriel
20 x1, y1, z1 = 1, 1, 1
21 x2, y2, z2 = -1, -1, 1
22 x3 = y1*z2 - z1*y2
23 y3 = z1*x2 - x1*z2
24 z3 = x1*y2 - y1*x2
25 print('produit vectoriel :',x1,y1,z1,x2,y2,z2,'->',x3,y3,z3)
26
27 # produit mixte
28 x1, y1, z1 = 1, 1, 1
29 x2, y2, z2 = -1, -1, 1
30 v = (y1*z2 - z1*y2)*x3 + (z1*x2 - x1*z2)*y3 + (x1*y2 - y1*x2)*z3
31 print('produit mixte :',x1,y1,z1,x2,y2,z2,'->',v)

```

TD 2.39 : PRIX D'UNE PHOTOCOPIE

```

1 # -*- coding: utf-8 -*-
2
3 n = 20
4 if n > 30 : prix = 10*0.1 + 20*0.09 + (n - 30)*0.08
5 elif n > 10 : prix = 10*0.1 + (n - 10)*0.09
6 else : prix = n*0.1
7 print(n,'->',prix)

```

TD 2.40 : CALCUL DES IMPÔTS

```

1 # -*- coding: utf-8 -*-
2
3 age = 25
4 sexe = 'f'
5 if age > 18 :
6     if sexe == 'm' : impot = True
7     elif age < 35 : impot = True
8     else : impot = False
9 else : impot = False
10 print(age,sexe,'->',impot)

```

TD 2.41 : DÉVELOPPEMENTS LIMITÉS

```

1 # -*- coding: utf-8 -*-
2

```

```

3 from math import *
4
5 x = 0.25
6
7 # développement limité 1 : sinh(x)
8 k = 0
9 u = x
10 y = u
11 s = 1.e-6
12 while fabs(u) > s :
13     u = u*x*x/((2*k+2)*(2*k+3))
14     y = y + u
15     k = k + 1
16 print('sinh    :',x,s,'->',y,':',sinh(x))
17
18 # développement limité 2 : cosh(x)
19 k = 0
20 u = 1.
21 y = u
22 s = 1.e-6
23 while fabs(u) > s :
24     u = u*x*x/((2*k+1)*(2*k+2))
25     y = y + u
26     k = k + 1
27 print('cosh    :',x,s,'->',y,':',cosh(x))
28
29 # développement limité 3 : cos(x)
30 k = 0
31 u = 1.
32 y = u
33 s = 1.e-6
34 while fabs(u) > s :
35     u = -u*x*x/((2*k+1)*(2*k+2))
36     y = y + u
37     k = k + 1
38 print('cos     :',x,s,'->',y,':',cos(x))
39
40 # développement limité 4 : log(1+x)
41 if fabs(x) < 1 :
42     k = 0
43     u = x
44     y = u
45     s = 1.e-6
46     while fabs(u) > s :
47         u = -u*x*(k+1)/(k+2)
48         y = y + u
49         k = k + 1
50 print('log     :',x,s,'->',y,':',log(1+x))
51
52 # développement limité 5 : arctan(x)
53 if fabs(x) < 1 :
54     k = 0
55     u = x
56     y = u
57     s = 1.e-6
58     while fabs(u) > s :
59         u = -u*x*x*(2*k+1)/(2*k+3)
60         y = y + u
61         k = k + 1

```

```
62 print('arctan :',x,s,'->',y,':',atan(x))
```

TD 2.42 : TABLES DE VÉRITÉ

```
1 # -*- coding: utf-8 -*-
2
3 # circuit 1
4 for a in [0,1] :
5     for b in [0,1] :
6         for c in [0,1] :
7             s = (a and not b) or (b and not a)
8             print(a, b, c, ':', int(s))
9 print()
10
11 # circuit 2
12 for a in [0,1] :
13     for b in [0,1] :
14         s = not (not (a and not b) and not (b and not a))
15         print(a, b, ':', int(s))
16 print()
17
18 # circuit 3
19 for a in [0,1] :
20     for b in [0,1] :
21         s = (a and not b) or (not a and b)
22         t = (not a) and b
23         print(a, b, ':', int(s), int(t))
24 print()
25
26 # circuit 4
27 for a in [0,1] :
28     for b in [0,1] :
29         for c in [0,1] :
30             s = (a != (b != c))
31             t = (b and c) or (a and (b != c))
32             print(a, b, c, ':', int(s), int(t))
33 print()
34
35 # circuit 5
36 for a in [0,1] :
37     for b in [0,1] :
38         for c in [0,1] :
39             s = not (not (a and b) and not (a and c) and not (b and c))
40             print(a, b, c, ':', int(s))
41 print()
42
43 # circuit 6
44 for a in [0,1] :
45     for b in [0,1] :
46         for c in [0,1] :
47             u = not a and not b and not c
48             v = a and not b and not c
49             w = a and b and not c
50             s = not (not u and not v and not w)
51             print(a, b, c, ':', int(s))
52 print()
53
54 # circuit 7
```

```

55 for a in [0,1] :
56     for b in [0,1] :
57         for c in [0,1] :
58             u = (b and not c) or (not b and c)
59             s = (a and not u) or (not a and u)
60             t = (b and c) or (a and u)
61             print(a, b, c, ': ', int(u), int(s), int(t))
62 print()
63
64 # circuit 8
65 for a in [0,1] :
66     for b in [0,1] :
67         for c in [0,1] :
68             s7 = a and b and c
69             s6 = a and b and not c
70             s5 = a and not b and c
71             s4 = a and not b and not c
72             s3 = not a and b and c
73             s2 = not a and b and not c
74             s1 = not a and not b and c
75             s0 = not a and not b and not c
76             print(a, b, c, ': ', int(s0), int(s1), int(s2), int(s3), int(s4), int(s5), int(s6), int(s7))

```

TD 2.43 : DESSINS GÉOMÉTRIQUES

```

1 # -*- coding: utf-8 -*-
2
3 from turtle import *
4
5 # dessin 1 : étoile à 5 branches
6 forward(20)
7 right(144)
8 forward(20)
9 right(144)
10 forward(20)
11 right(144)
12 forward(20)
13 right(144)
14 forward(20)
15 right(144)
16
17 # dessin 2 : losange
18 forward(10)
19 left(45)
20 forward(10)
21 left(135)
22 forward(10)
23 left(45)
24 forward(10)
25 left(135)

```

TD 2.44 : POLICE D'ASSURANCE

```

1 # -*- coding: utf-8 -*-
2
3 # assurance à points

```

```

4 age = 23
5 anciennete = 3
6 fidelite = 1
7 n = 0
8
9 pts = 0
10 if age < 25 : pts = pts + 1 # jeune
11 if anciennete < 2 : pts = pts + 1 # jeune conducteur
12 pts = pts + n # accidents
13 if pts < 3 and fidelite > 1 : pts = pts - 1 # fidélité
14 if pts == -1 : print('tarif A')
15 elif pts == 0 : print('tarif B')
16 elif pts == 1 : print('tarif C')
17 elif pts == 2 : print('tarif D')
18 else : print('refus d\'assurer')

```

TD 2.45 : ZÉRO D'UNE FONCTION

```

1 # -*- coding: utf-8 -*-
2
3 from math import *
4
5 # zéro d'une fonction : méthode par dichotomie
6 x1 = 1.
7 x2 = 2.
8 s = 1.e-9
9 f = cos
10
11 while (x2 - x1) > s :
12     x = (x1 + x2)/2.
13     if f(x1)*f(x) < 0 : x2 = x
14     else : x1 = x
15 print(x, cos(x))
16
17
18 # zéro d'une fonction : méthode des tangentes
19 x1 = 1.
20 x2 = 2.
21 s = 1.e-9
22 f = cos
23
24 df = sin
25 x = x2 - f(x2)/(-df(x2))
26 while fabs(x-x2) > s :
27     x2 = x
28     x = x - f(x)/(-df(x))
29 print(x, cos(x))
30
31 # zéro d'une fonction : méthode des sécantes
32 x1 = 1.
33 x2 = 2.
34 s = 1.e-9
35 f = cos
36
37 df = (f(x2)-f(x1))/(x2-x1)
38 x = x2 - f(x2)/df
39 while fabs(x-x2) > s :
40     x2 = x

```

```
41     df = (f(x2)-f(x1))/(x2-x1)
42     x = x - f(x)/df
43 print(x, cos(x))
44
45 # zéro d'une fonction : méthode des cordes
46 x1 = 1.
47 x2 = 2.
48 s = 1.e-9
49 f = cos
50 while (x2-x1) > s :
51     x = (x2*f(x1) - x1*f(x2))/(f(x1)-f(x2))
52     if f(x1)*f(x) < 0 : x2 = x
53     else : x1 = x
54 print(x, cos(x))
55
```

Chapitre 3

Procédures et fonctions

TD 3.1 : CODAGE DES ENTIERS POSITIFS (1)

```
1 # -*- coding: utf-8 -*-
2
3 n = 83
4 b = 8
5 k = 8
6
7 code = []
8 quotient = n
9 for i in range(k): code.append(0)
10
11 i = k - 1
12 while quotient != 0 and i >= 0:
13     code[i] = quotient%b
14     quotient = quotient//b
15     i = i - 1
16
17 print(n,b,k,'->',code)
```

TD 3.2 : CODAGE D'UN NOMBRE FRACTIONNAIRE

```
1 # -*- coding: utf-8 -*-
2
3 x = 140.8125
4
5 # signe
6 s = int(x < 0)
7 cs = [s]
8 print('signe :',x,s,cs)
9
10 # partie entière
11 pe = int(abs(x))
12
13 b = 2
14 cpe = []
15 quotient = pe
16 while quotient != 0:
17     cpe.insert(0,quotient%b)
18     quotient = quotient//b
19 print('partie entière :',x,pe,cpe)
20
```

```

21 # partie fractionnaire
22 pf = abs(x) - pe
23
24 b = 2
25 cpf = []
26 fraction = pf
27 while fraction != 0:
28     cpf.append(int(fraction*b))
29     fraction = fraction*b - int(fraction*b)
30 print('partie fractionnaire :',x,pf,cpf)
31
32 # mantisse
33 mantisse = cpe + cpf
34 print('mantisse :',x,mantisse)
35
36 # exposant
37 exposant = len(cpe)-1
38
39 b = 2
40 cexposant = []
41 quotient = exposant
42 while quotient != 0:
43     cexposant.insert(0,quotient%b)
44     quotient = quotient//b
45 print('exposant :',x,exposant,cexposant)
46
47 # affichage
48 print(x, ' -> ', '(-1)**',s,sep='',end='')
49 print('*(',mantisse[0],'.',sep='',end='')
50 for i in range(1,len(mantisse)): print(mantisse[i],sep='',end='')
51 print(')*',b,'**(',sep='',end='')
52 for i in range(0,len(cexposant)): print(cexposant[i],sep='',end='')
53 print(')')

```

TD 3.3 : DÉCODAGE BASE B → DÉCIMAL

```

1 # -*- coding: utf-8 -*-
2
3 def decodage(code,b):
4     """
5     n = decodage(code,b)
6     n est l'entier décimal correspondant au code (liste de chiffres)
7     dans la base b
8
9     >>> decodage([1,2,3],5)
10    38
11    >>> decodage([1,2,3],8)
12    83
13    >>> decodage([1,2,3],16)
14    291
15    >>> decodage([1,2,3],10)
16    123
17    >>> decodage([1,0,11],12)
18    155
19    """
20    assert type(code) is list
21    assert type(b) is int and b > 1
22

```



```

23     n = 0
24     for i in range(len(code)):
25         n = n + code[i]*b**(len(code)-1-i)
26
27     return n
28
29 #-----
30 if __name__ == "__main__":
31     import doctest
32     doctest.testmod()

```

TD 3.4 : CODAGE DES ENTIERS POSITIFS (2)

```

1 # -*- coding: utf-8 -*-
2
3 def codage(n,b,k):
4     """
5     code = codage(n,b,k)
6     code en base b sur k bits de l'entier décimal n
7
8     >>> codage(23,2,8)
9     [0, 0, 0, 1, 0, 1, 1, 1]
10    >>> codage(23,5,3)
11    [0, 4, 3]
12    >>> codage(23,21,3)
13    [0, 1, 2]
14    >>> codage(23,25,2)
15    [0, 23]
16    """
17    assert type(n) is int
18    assert type(b) is int
19    assert type(k) is int
20    assert n >= 0 and b > 1 and k > 0
21    assert n < b**k - 1
22
23    code = []
24    quotient = n
25    for i in range(k): code.append(0)
26
27    i = k - 1
28    while quotient != 0 and i >= 0:
29        code[i] = quotient%b
30        quotient = quotient//b
31        i = i - 1
32
33    return code
34
35
36 #-----
37 if __name__ == "__main__":
38     import doctest
39     doctest.testmod()

```

TD 3.5 : UNE SPÉCIFICATION, DES IMPLÉMENTATIONS

```

1 # -*- coding: utf-8 -*-
2

```

```

3 # suite géométrique 1 : version constante
4 # complexité indépendante de n
5
6 def sommeGeometrique1(n,a,b) :
7     """
8     s = sommeGeometrique1(n,a,b)
9     somme des n premiers termes d'une suite géométrique
10    uk = a*b**k
11
12    >>> sommeGeometrique1(5,1,2)
13    63
14    >>> sommeGeometrique1(5,1,3)
15    364
16    """
17    assert type(a) is int and a != 0
18    assert type(b) is int and b != 1
19    assert type(n) is int and n >= 0
20
21    s = a*(b**(n+1) - 1)/(b-1)
22
23    return s
24
25 # suite géométrique 2 : version itérative
26 # complexité linéaire O(n)
27
28 def sommeGeometrique2(n,a,b) :
29     """
30     s = sommeGeometrique2(n,a,b)
31     somme des n premiers termes d'une suite géométrique
32     uk = a*b**k
33
34     >>> sommeGeometrique2(5,1,2)
35     63
36     >>> sommeGeometrique2(5,1,3)
37     364
38     """
39     assert type(a) is int and a != 0
40     assert type(b) is int and b != 1
41     assert type(n) is int and n >= 0
42
43     s = 0
44     for i in range(n+1):
45         s = s + a*b**i
46
47     return s
48
49 #-----
50 if __name__ == "__main__":
51     import doctest
52     doctest.testmod()

```

TD 3.6 : PASSAGE PAR VALEUR

```

1 # -*- coding: utf-8 -*-
2
3 # swap : version 1
4 x, y = 1, 2
5 print('1 avant :',x,y)

```

```

6
7 tmp = x
8 x = y
9 y = tmp
10 print('1 après :',x,y,' il y a permutation des originaux')
11
12 # swap : version 2
13 def swap(x,y):
14     tmp = x
15     x = y
16     y = tmp
17     print('2 pendant :',x,y,' il y a bien permutation des copies')
18     return
19
20 x, y = 1, 2
21 print('2 avant :',x,y)
22
23 swap(x,y)
24 print('2 après :',x,y,' mais pas des originaux')

```

TD 3.7 : VALEURS PAR DÉFAUT

```

1 #-*- coding: utf-8 -*-
2
3 def codage(n,b=2,k=8):
4     """
5     code = codage(n,b,k)
6     code en base b sur k bits de l'entier décimal n
7
8     >>> codage(23)
9     [0, 0, 0, 1, 0, 1, 1, 1]
10    >>> codage(23,2)
11    [0, 0, 0, 1, 0, 1, 1, 1]
12    >>> codage(23,2,8)
13    [0, 0, 0, 1, 0, 1, 1, 1]
14    >>> codage(23,5)
15    [0, 0, 0, 0, 0, 0, 4, 3]
16    >>> codage(23,5,3)
17    [0, 4, 3]
18    >>> codage(23,21,3)
19    [0, 1, 2]
20    >>> codage(23,25,2)
21    [0, 23]
22    """
23    assert type(n) is int
24    assert type(b) is int
25    assert type(k) is int
26    assert n >= 0 and b > 1 and k > 0
27    assert n < b**k - 1
28
29    code = []
30    quotient = n
31    for i in range(k): code.append(0)
32
33    i = k - 1
34    while quotient != 0 and i >= 0:
35        code[i] = quotient%b
36        quotient = quotient//b

```

```

37         i = i - 1
38
39     return code
40
41
42 #-----
43 if __name__ == "__main__":
44     import doctest
45     doctest.testmod()

```

TD 3.8 : PORTÉE DES VARIABLES

```

1  # -*- coding: utf-8 -*-
2
3  def f(x):
4      x = 2*x
5      print('f', x)
6      return x
7
8  def g(x):
9      x = 2*f(x)
10     print('g', x)
11     return x
12
13  def h(x):
14      x = 2*g(f(x))
15      print('h', x)
16      return x
17
18  x = 5
19  print(x)
20  print('----')
21  y = f(x)
22  print(x)
23  print('----')
24  z = g(x)
25  print(x)
26  print('----')
27  t = h(x)
28  print(x)
29  print('----')
30  print()
31
32  x = 5
33  print(x)
34  print('----')
35  x = f(x)
36  print(x)
37  print('----')
38  x = g(x)
39  print(x)
40  print('----')
41  x = h(x)
42  print(x)

```

TD 3.9 : TOURS DE HANOÏ à la main

```

1 # -*- coding: utf-8 -*-
2
3 def hanoi(n,a,b,c):
4     """
5     déplace n disques de la tour a à la tour c
6     en utilisant la tour b
7
8     >>> hanoi(1,'a','b','c')
9     disque 1 de a vers c
10    >>> hanoi(2,'a','b','c')
11    disque 1 de a vers b
12    disque 2 de a vers c
13    disque 1 de b vers c
14    >>> hanoi(3,'a','b','c')
15    disque 1 de a vers c
16    disque 2 de a vers b
17    disque 1 de c vers b
18    disque 3 de a vers c
19    disque 1 de b vers a
20    disque 2 de b vers c
21    disque 1 de a vers c
22    >>> hanoi(4,'a','b','c')
23    disque 1 de a vers b
24    disque 2 de a vers c
25    disque 1 de b vers c
26    disque 3 de a vers b
27    disque 1 de c vers a
28    disque 2 de c vers b
29    disque 1 de a vers b
30    disque 4 de a vers c
31    disque 1 de b vers c
32    disque 2 de b vers a
33    disque 1 de c vers a
34    disque 3 de b vers c
35    disque 1 de a vers b
36    disque 2 de a vers c
37    disque 1 de b vers c
38    """
39    assert type(n) is int and n >= 0
40
41    if n > 0:
42        hanoi(n-1,a,c,b)
43        print('disque',n,'de',a,'vers',c)
44        hanoi(n-1,b,a,c)
45    else: pass
46    return
47
48 #-----
49 if __name__ == "__main__":
50     import doctest
51     doctest.testmod()

```

TD 3.10 : PGCD ET PPCM DE 2 ENTIERS (1)

```

1 # -*- coding: utf-8 -*-
2
3 # pgcd

```

```

4 def pgcd(a,b):
5     """
6     d = pgcd(a,b)
7     plus grand commun diviseur des 2 entiers a et b
8
9     >>> pgcd(12,18)
10    6
11    >>> pgcd(18,12)
12    6
13    >>> pgcd(21,15)
14    3
15    >>> pgcd(5,7)
16    1
17    """
18    assert type(a) is int and a >= 0
19    assert type(b) is int and b >= 0
20
21    if b == 0:
22        return a
23    else:
24        return pgcd(b,a%b)
25
26 # ppcm
27 def ppcm(a,b):
28     """
29     m = ppcm(a,b)
30     plus petit commun multiple de 2 entiers a et b
31
32     >>> ppcm(12,18)
33     36
34     >>> ppcm(18,12)
35     36
36     >>> ppcm(21,15)
37     105
38     >>> ppcm(5,7)
39     35
40     >>> ppcm(0,6)
41     0
42     """
43     assert type(a) is int and a >= 0
44     assert type(b) is int and b >= 0
45
46     if a == 0 or b == 0:
47         return 0
48     else:
49         return a*b//pgcd(a,b)
50
51 #-----
52 if __name__ == "__main__":
53     import doctest
54     doctest.testmod()

```

TD 3.11 : SOMME ARITHMÉTIQUE

```

1 # -*- coding: utf-8 -*-
2
3 # somme arithmétique 1 : version constante
4 def sommeArithmetique1(a,b,n):

```

```

5     """
6     s = sommeArithmetique1(a,b,n)
7     somme des n premiers termes d'une suite arithmétique
8     uk = a + b*k
9
10    >>> sommeArithmetique1(0,1,5)
11    15
12    >>> sommeArithmetique1(0,1,6)
13    21
14    >>> sommeArithmetique1(1,1,6)
15    28
16    >>> sommeArithmetique1(1,2,6)
17    49
18    """
19    assert type(a) is int and a >= 0
20    assert type(b) is int and b >= 0
21    assert type(n) is int and n >= 0
22
23    s = a*(n+1) + b*n*(n+1)//2
24
25    return s
26
27
28 # somme arithmétique 2 : version itérative
29 def sommeArithmetique2(a,b,n):
30     """
31     s = sommeArithmetique2(a,b,n)
32     somme des n premiers termes d'une suite arithmétique
33     uk = a + b*k
34
35     >>> sommeArithmetique2(0,1,5)
36     15
37     >>> sommeArithmetique2(0,1,6)
38     21
39     >>> sommeArithmetique2(1,1,6)
40     28
41     >>> sommeArithmetique2(1,2,6)
42     49
43     """
44     assert type(a) is int and a >= 0
45     assert type(b) is int and b >= 0
46     assert type(n) is int and n >= 0
47
48     s = 0
49     for i in range(n+1):
50         s = s + a + b*i
51
52     return s
53
54 # somme arithmétique 3 : version récursive
55 def sommeArithmetique3(a,b,n):
56     """
57     s = sommeArithmetique3(a,b,n)
58     somme des n premiers termes d'une suite arithmétique
59     uk = a + b*k
60
61     >>> sommeArithmetique3(0,1,5)
62     15
63     >>> sommeArithmetique3(0,1,6)

```

```

64     21
65     >>> sommeArithmetique3(1,1,6)
66     28
67     >>> sommeArithmetique3(1,2,6)
68     49
69     """
70     assert type(a) is int and a >= 0
71     assert type(b) is int and b >= 0
72     assert type(n) is int and n >= 0
73
74     if n == 0:
75         return a
76     else:
77         return (a + b*n) + sommeArithmetique3(a,b,n-1)
78
79 #-----
80 if __name__ == "__main__":
81     import doctest
82     doctest.testmod()

```

TD 3.12 : COURBES FRACTALES

```

1  # -*- coding: utf-8 -*-
2
3  from turtle import *
4
5  def draw(n,d):
6      assert type(n) is int
7      assert n >= 0
8
9      if n == 0: forward(d)
10     else:
11         draw(n-1,d/3.)
12         left(60)
13         draw(n-1,d/3.)
14         right(120)
15         draw(n-1,d/3.)
16         left(60)
17         draw(n-1,d/3.)
18     return
19
20 up()
21 goto(0,-300)
22 setheading(0)
23 down()
24 draw(0,900)
25
26 up()
27 goto(0,-275)
28 setheading(0)
29 down()
30 draw(1,900)
31
32 up()
33 goto(0,-200)
34 setheading(0)
35 down()
36 draw(2,900)

```



```

37
38 up()
39 goto(0,-25)
40 setheading(0)
41 down()
42 draw(3,900)
43
44 up()
45 goto(0,150)
46 setheading(0)
47 down()
48 draw(4,900)

```

TD 3.13 : QCM (3)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Les bonnes réponses sont extraites directement des notes de cours :
5 1c, 2a, 3c, 4a, 5c, 6a, 7c, 8a, 9c, 10d, 11d
6 """

```

TD 3.14 : PASSAGE DES PARAMÈTRES

```

1 # -*- coding: utf-8 -*-
2
3 def f(x):
4     y = x + 2
5     return y
6
7 def g(z):
8     v = 2*f(z)
9     return v
10
11 def h(a):
12     b = g(f(a))
13     return b
14
15 # appels équivalents
16 t = 2
17 # u = f(t)
18 x = t
19 y = x + 2
20 tmp = y
21 del x, y
22 u = tmp
23 del tmp
24 print(t,u,f(t))
25
26 # u = g(t)
27 z = t
28 x = z
29 y = x + 2
30 tmp = y
31 del x, y
32 v = 2*tmp

```

```

33 tmp = v
34 del v, z
35 u = tmp
36 del tmp
37 print(t,u,g(t))
38
39 # u = h(t)
40 a = t
41 x = a
42 y = x + 2
43 tmp = y
44 del x, y
45 z = tmp
46 del tmp
47 x = z
48 y = x + 2
49 tmp = y
50 del x, y
51 v = 2*tmp
52 tmp = v
53 del v, z
54 b = tmp
55 del tmp
56 tmp = b
57 del a, b
58 u = tmp
59 del tmp
60 print(t,u,h(t))

```

TD 3.15 : PORTÉE DES VARIABLES (2)

```

1 # -*- coding: utf-8 -*-
2
3 def f(x):
4     x = x + 2
5     print('f', x)
6     return x
7
8 def g(x):
9     x = 2*f(x)
10    print('g', x)
11    return x
12
13 def h(x):
14    x = g(f(x))
15    print('h', x)
16    return x
17
18 x = 5
19 print(x)
20 print('---')
21 x = x + 2
22 print(x)
23 print('---')
24 x = 2 * (x + 2)
25 print(x)
26 print('---')
27 print()

```

```

28
29 x = 5
30 print(x)
31 print('----')
32 y = f(x)
33 print(x, y)
34 print('----')
35 z = 2*f(y)
36 print(x, y, z)
37 print('----')
38 print()
39
40 x = 5
41 print(x)
42 print('----')
43 z = 2*f(f(x))
44 print(x, z)
45 print('----')
46 print()
47
48 x = 5
49 print(x)
50 print('----')
51 f(x)
52 print('----')
53 print(x)
54 print('----')
55 g(x)
56 print('----')
57 print(x)
58 print('----')
59 h(x)
60 print('----')
61 print(x)
62 print('----')
63 print()

```

TD 3.16 : SUITE GÉOMÉTRIQUE

```

1 # -*- coding: utf-8 -*-
2
3 def sommeGeometrique(n,a,b):
4     """
5     s = sommeGeometrique1(n,a,b)
6     somme des n premiers termes d'une suite géométrique
7     uk = a*b**k
8
9     >>> sommeGeometrique(5,1,2)
10    63
11    >>> sommeGeometrique(5,1,3)
12    364
13    """
14    assert type(a) is int and a != 0
15    assert type(b) is int and b != 1
16    assert type(n) is int and n >= 0
17
18    if n == 0:
19        return 1

```

```

20     else:
21         return sommeGeometrique(n-1,a,b) + (a * b**n)
22
23 #-----
24 if __name__ == "__main__":
25     import doctest
26     doctest.testmod()

```

TD 3.17 : PUISSANCE ENTIÈRE

```

1 # -*- coding: utf-8 -*-
2
3 def puissance(x,n):
4     """
5     y = puissance(x,n)
6     est la puissance entière de x de degré n
7
8     >>> puissance(3,2)
9     9
10    >>> puissance(2,3)
11    8
12    >>> puissance(4,0)
13    1
14    """
15    assert type(n) is int and n >= 0
16    if n == 0:
17        return 1
18    else:
19        return x*puissance(x,n-1)
20
21 #-----
22 if __name__ == "__main__":
23     import doctest
24     doctest.testmod()

```

TD 3.18 : COEFFICIENTS DU BINÔME

```

1 # -*- coding: utf-8 -*-
2
3 def coefficientBinome(n,p):
4     """
5     c = coefficientBinome(n,p)
6     p-ième coefficient du binôme (a+b)**n
7
8     >>> n = 6
9     >>> for i in range(0,n+1):
10         for p in range(0,i+1):
11             print(coefficientBinome(i,p),end=' ')
12         print()
13     ...
14     1
15     1 1
16     1 2 1
17     1 3 3 1
18     1 4 6 4 1
19     1 5 10 10 5 1

```

```

20     1 6 15 20 15 6 1
21     """
22     assert type(n) is int and type(p) is int
23     assert n >= 0 and p >= 0 and p <= n
24
25     if p == 0 or n == 0 or n == p:
26         return 1
27     else:
28         return coefficientBinome(n-1,p) + coefficientBinome(n-1,p-1)
29
30
31 #-----
32 if __name__ == "__main__":
33     import doctest
34     doctest.testmod()

```

TD 3.19 : FONCTION D'ACKERMAN

```

1 # -*- coding: utf-8 -*-
2
3 def ackerman(m,n):
4     """
5     y = ackerman(m,n)
6     calcul la fonction d'Ackerman pour le couple d'entiers m,n
7
8     >>> ackerman(0,0)
9     1
10    >>> ackerman(1,0)
11    2
12    >>> ackerman(2,0)
13    3
14    >>> ackerman(0,1)
15    2
16    >>> ackerman(0,2)
17    3
18    >>> ackerman(1,1)
19    3
20    >>> ackerman(1,2)
21    4
22    >>> ackerman(2,2)
23    7
24    >>> ackerman(3,3)
25    61
26    """
27    assert type(n) is int and type(m) is int
28    assert n >= 0 and m >= 0
29
30    if m == 0:
31        return n + 1
32    elif n == 0:
33        return ackerman(m-1,1)
34    else:
35        return ackerman(m-1,ackerman(m,n-1))
36
37 #-----
38 if __name__ == "__main__":
39     import doctest
40     doctest.testmod()

```

TD 3.20 : ADDITION BINAIRE

```

1 # -*- coding: utf-8 -*-
2
3 def add2(code1,code2):
4     """
5     sum2 = add2(code1,code2)
6     addition binaire code1 + code2
7
8     >>> add2([1,0,1],[1])
9     [1, 1, 0]
10    >>> add2([1,0,1],[1,0])
11    [1, 1, 1]
12    >>> add2([1,0],[1,0,1])
13    [1, 1, 1]
14    >>> add2([1,0,1],[1,1])
15    [1, 0, 0, 0]
16    """
17    assert type(code1) is list
18    assert type(code2) is list
19
20    sum2 = []
21    diffLen = len(code1) - len(code2)
22    if diffLen > 0:
23        for i in range(diffLen): code2.insert(0,0)
24    else:
25        for i in range(-diffLen): code1.insert(0,0)
26
27    for i in range(len(code1)): sum2.append(0)
28
29    carry = 0
30    for i in range(len(code1)-1,-1,-1):
31        value = code1[i] + code2[i] + carry
32        if value >= 2:
33            sum2[i] = value - 2
34            carry = 1
35        else:
36            sum2[i] = value
37            carry = 0
38
39    if carry == 1: sum2.insert(0,1)
40
41    return sum2
42
43 #-----
44 if __name__ == "__main__":
45     import doctest
46     doctest.testmod()

```

TD 3.21 : COMPLÉMENT À 2

```

1 # -*- coding: utf-8 -*-
2
3 def neg2(code):
4     """
5     neg = neg2(code)

```

```

6    complément à 2 d'un entier binaire
7
8    >>> neg2([0,0,0,1,0,1,1,1])
9    [1, 1, 1, 0, 1, 0, 0, 1]
10   >>> neg2([1, 1, 1, 0, 1, 0, 0, 1])
11   [0, 0, 0, 1, 0, 1, 1, 1]
12
13   >>> for a in [0,1]:
14   ...     for b in [0,1]:
15   ...         for c in [0,1]:
16   ...             add2([a,b,c],neg2([a,b,c]))
17   [0, 0, 0]
18   [1, 0, 0, 0]
19   [1, 0, 0, 0]
20   [1, 0, 0, 0]
21   [1, 0, 0, 0]
22   [1, 0, 0, 0]
23   [1, 0, 0, 0]
24   [1, 0, 0, 0]
25   """
26
27   assert type(code) is list
28   neg = []
29
30   carry = 1
31   for i in range(len(code)): neg.append(int(not code[i]))
32   for i in range(len(code)):
33       value = neg[len(code)-1-i] + carry
34       if value >= 2:
35           neg[len(code)-1-i] = value - 2
36           carry = 1
37       else:
38           neg[len(code)-1-i] = value
39           carry = 0
40
41   return neg
42
43 #-----
44 def add2(code1,code2):
45     """
46     sum2 = add2(code1,code2)
47     addition binaire code1 + code2
48
49     >>> add2([1,0,1],[1])
50     [1, 1, 0]
51     >>> add2([1,0,1],[1,0])
52     [1, 1, 1]
53     >>> add2([1,0],[1,0,1])
54     [1, 1, 1]
55     >>> add2([1,0,1],[1,1])
56     [1, 0, 0, 0]
57     """
58     assert type(code1) is list
59     assert type(code2) is list
60
61     sum2 = []
62     diffLen = len(code1) - len(code2)
63     if diffLen > 0:
64         for i in range(diffLen): code2.insert(0,0)

```

```

65     else:
66         for i in range(-diffLen): code1.insert(0,0)
67
68     for i in range(len(code1)): sum2.append(0)
69
70     carry = 0
71     for i in range(len(code1)-1,-1,-1):
72         value = code1[i] + code2[i] + carry
73         if value >= 2:
74             sum2[i] = value - 2
75             carry = 1
76         else:
77             sum2[i] = value
78             carry = 0
79
80     if carry == 1: sum2.insert(0,1)
81
82     return sum2
83
84 #-----
85 if __name__ == "__main__":
86     import doctest
87     doctest.testmod()

```

TD 3.22 : CODAGE-DÉCODAGE DES RÉELS

(on pourra vérifier sur le site <http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html>)

```

1  # -*- coding: utf-8 -*-
2
3  def decodeIeee754(code):
4      """
5      x = decodeIeee754(code)
6      valeur décimale du réel correspondant au code IEEE 754
7
8      >>> decodeIeee754(ieee754(-31.7,2)[1])
9      -31.7
10     >>> decodeIeee754(ieee754(-31.7e-5,2)[1])
11     -0.000317
12     >>> decodeIeee754(ieee754(-31.7e5,2)[1])
13     -3170000.0
14     """
15     assert type(code) is list
16     assert len(code) == 32 or len(code) == 64
17
18     ke, km, kieee, biais = precision754(len(code)//32)
19
20     signe = (-1)**code[0]
21
22     cexposant = code[1:ke+1]
23     cmantisse = code[ke+1:km+ke+1]
24
25     exposant = decodage(cexposant,2) - biais
26
27     mantisse = 1
28     for i in range(len(cmantisse)):
29         mantisse = mantisse + cmantisse[i]*2**(-i-1)
30
31     x = signe*mantisse*2**exposant

```



```

32
33     return x
34
35 #-----
36 def ieee754(x,precision=1) :
37     """
38     (statut, code) = ieee754(x,precision)
39     codage selon la norme IEEE 754 du réel x en simple précision
40     (precision == 1) ou en double précision (precision == 2).
41     Si statut == 'normal', code est le code IEEE754 recherché
42     sinon statut = 'underflow' ou 'overflow' et code = [0]
43
44     >>> ieee754(0.0,1)
45     ('normal', [0,\
46 0, 0, 0, 0, 0, 0, 0, 0, \
47 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
48     >>> ieee754(-0.0625,1)
49     ('normal', [1,\
50 0, 1, 1, 1, 1, 0, 1, 1, \
51 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
52     >>> ieee754(-0.0625e-250,1)
53     ('underflow', [1,\
54 0, 0, 0, 0, 0, 0, 0, 0, \
55 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
56     >>> ieee754(-0.0625e250,1)
57     ('overflow', [1,\
58 1, 1, 1, 1, 1, 1, 1, 1, \
59 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
60     >>> ieee754(0.0625,2)
61     ('normal', [0,\
62 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, \
63 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
64 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
65     >>> ieee754(-0.0625e-250,2)
66     ('normal', [1,\
67 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, \
68 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, \
69 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0])
70     >>> ieee754(173.2679,1)
71     ('normal', [0,\
72 1, 0, 0, 0, 0, 1, 1, 0, \
73 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1])
74     >>> ieee754(173.2679,2)
75     ('normal', [0,\
76 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, \
77 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, \
78 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0])
79     """
80     assert type(x) is float
81     assert precision in [1,2]
82
83     ke, km, kieee, biais = precision754(precision)
84
85     statut = 'normal'
86     code = [0 for i in range(kieee)]
87     if x != 0.0 :
88         statut, mantisse, exposant = mantisse_exposant(x,precision)
89         code[0] = signe(x)
90         code[1:ke+2] = exposant

```

```

91         code[ke+1:km+ke+1] = mantisse
92
93     return statut, code
94
95 #-----
96 def precision754(precision) :
97     assert precision in [1,2]
98
99     if precision == 1 :
100         ke, km, kieee, bias = 8, 23, 32, 127
101     else :
102         ke, km, kieee, bias = 11, 52, 64, 1023
103
104     return ke, km, kieee, bias
105
106 #-----
107 def mantisse_exposant(x,precision=1) :
108     assert type(x) is float
109     assert precision in [1,2]
110
111     ke, km, kieee, bias = precision754(precision)
112     cbias = codage(bias,2,ke,True)[1]
113
114     pe, pf = partieEntiere(x), partieFractionnaire(x)
115
116     expo, debut = 0, False
117     statut, mantisse = codage(pe,2,km+1)
118
119     if statut == 'overflow' :
120         exposant = codage(2**ke-1,2,ke,True)[1]
121         mantisse = codage(0,2,km,True)[1]
122     else :
123         if mantisse == [0] :
124             debut = True
125             mantisse = []
126
127         expo = len(mantisse)-1
128         i = len(mantisse)
129         while (pf != 0) and (i < km+1):
130             pf = pf * 2
131             pe = int(pf)
132             pf = pf - pe
133             if (pe == 0) and debut == True :
134                 expo = expo - 1
135             else:
136                 debut = False
137                 mantisse.append(pe)
138                 i = i + 1
139
140         if len(mantisse) > 0 and mantisse[0] == 1 :
141             del mantisse[0]
142
143         for i in range(len(mantisse),km):
144             mantisse.append(0)
145
146         if expo > bias :
147             statut = 'overflow'
148             expo = 2**ke-1
149             mantisse = codage(0,2,km,True)[1]

```

```

150         elif expo <= -bias :
151             statut = 'underflow'
152             expo = -bias
153             mantisse = codage(0,2,km,True)[1]
154         else :
155             statut = 'normal'
156
157         expo = expo + bias
158         exposant = codage(expo,2,ke,True)[1]
159
160     return statut, mantisse, exposant
161
162 #-----
163 def signe(x) :
164     """
165     s = signe(x)
166     signe du réel x, 0 si x > 0, 1 si x < 0
167     >>> signe(5.7)
168     0
169     >>> signe(-5.7)
170     1
171     """
172     assert type(x) is float
173
174     return int(x < 0)
175
176 #-----
177 def partieEntiere(x) :
178     """
179     pe = partieEntiere(x)
180     partie entière du réel x
181
182     >>> partieEntiere(0.67)
183     0
184     >>> partieEntiere(-0.67)
185     0
186     >>> partieEntiere(34.8)
187     34
188     """
189     assert type(x) is float
190
191     return int(abs(x))
192
193 #-----
194 def partieFractionnaire(x) :
195     """
196     pf = partieFractionnaire(x)
197     partie fractionnaire du réel x
198
199     >>> partieFractionnaire(8.5)
200     0.5
201     >>> partieFractionnaire(-8.5)
202     0.5
203     >>> partieFractionnaire(0.5)
204     0.5
205     """
206     assert type(x) is float
207
208     return abs(x) - partieEntiere(x)

```

```

209
210 #-----
211 def codage(n,b,k,remplir=False) :
212     """
213     statut, code = codage(n,b,k,remplir)
214     code en base b sur k bits maximum l'entier décimal n
215     statut = 'normal' si 0 <= n < b**k, 'overflow' sinon
216     si remplir == True, on remplit à gauche de 0 pour avoir len(code) = k
217
218     >>> codage(0,2,8,False)
219     ('normal', [0])
220     >>> codage(0,2,8,True)
221     ('normal', [0, 0, 0, 0, 0, 0, 0, 0])
222     >>> codage(256,2,8,False)
223     ('overflow', [0])
224     >>> codage(65,2,8,False)
225     ('normal', [1, 0, 0, 0, 0, 0, 0, 1])
226     >>> codage(65,2,8,True)
227     ('normal', [0, 1, 0, 0, 0, 0, 0, 1])
228     >>> codage(65,5,4,True)
229     ('normal', [0, 2, 3, 0])
230     >>> codage(79,16,4,True)
231     ('normal', [0, 0, 4, 15])
232     """
233     assert type(n) is int and n >= 0
234     assert type(b) is int and b > 1
235     assert type(k) is int and k > 0
236
237     if n == 0 :
238         statut = 'normal'
239         code = [0]
240     elif n >= b**k :
241         statut = 'overflow'
242         code = [0]
243     else :
244         statut = 'normal'
245         code = []
246         i = 0
247         while (n != 0) and (i < k) :
248             code.insert(0,n%b)
249             n = n//b
250             i = i + 1
251
252     if remplir == True :
253         diffLen = k - len(code)
254         for i in range(diffLen):
255             code.insert(0,0)
256
257     return statut, code
258
259 #-----
260 def decodage(code,b=2) :
261     """
262     n = decodage(code,b)
263     valeur décimale de l'entier correspondant au code en base b
264
265     >>> decodage(codage(0,2,8,False)[1],2)
266     0
267     >>> decodage(codage(65,2,8,True)[1],2)

```

```

268     65
269     >>> decodage(codage(79,16,4,True)[1],16)
270     79
271     """
272     assert type(code) is list
273     assert type(b) is int and b > 1
274
275     x = 0
276     for i in range(len(code)) :
277         c = code[len(code)-1-i]
278         x = x + c*b**i
279
280     return x
281
282 #-----
283 if __name__ == "__main__":
284     import doctest
285     doctest.testmod()

```

TD 3.23 : INTÉGRATION NUMÉRIQUE

```

1  # -*- coding: utf-8 -*-
2
3  from math import *
4
5  # intégration 1 : méthode des rectangles
6  def rectangle_integration(f,x1,x2,n):
7      """
8      intégration de f(x) entre x1 et x2
9      par la méthode des n rectangles
10
11      >>> fabs(rectangle_integration(sin,0.,2*pi,100000)) < 1.e-5
12      True
13      >>> fabs(rectangle_integration(sin,0.,pi,100000) - 2.) < 1.e-5
14      True
15      >>> fabs(rectangle_integration(sin,0.,pi/2,100000) - 1) < 1.e-5
16      True
17      >>> fabs(rectangle_integration(cos,0.,pi,100000)) < 1.e-5
18      True
19      >>> fabs(rectangle_integration(cos,-pi/2,pi/2,100000) - 2) < 1.e-5
20      True
21      """
22      assert type(x1) is float
23      assert type(x2) is float
24      assert x1 <= x2
25
26      integral = 0.0
27      width = (x2 - x1)/n
28      x = x1 + width/2
29      while x < x2:
30          integral = integral + f(x)
31          x = x + width
32      integral = integral*width
33      return integral
34
35  # intégration 2 : méthode des trapèzes
36  def trapezoid_integration(f,x1,x2,n):
37

```

```

38     """
39     intégration de f(x) entre x1 et x2
40     par la méthode des n trapèzes
41
42     >>> fabs(trapezoid_integration(sin,0.,2*pi,100000)) < 1.e-5
43     True
44     >>> fabs(trapezoid_integration(sin,0.,pi,100000) - 2.) < 1.e-5
45     True
46     >>> fabs(trapezoid_integration(sin,0.,pi/2,100000) - 1) < 1.e-4
47     True
48     >>> fabs(trapezoid_integration(cos,0.,pi,100000)) < 1.e-4
49     True
50     >>> fabs(trapezoid_integration(cos,-pi/2,pi/2,100000) - 2) < 1.e-5
51     True
52     """
53     assert type(n) is int
54     assert type(x1) is float
55     assert type(x2) is float
56     assert x1 <= x2
57
58     integral = (f(x1) + f(x2))/2
59     width = (x2 - x1)/n
60     x = x1 + width
61     while x < x2:
62         integral = integral + f(x)
63         x = x + width
64     integral = integral*width
65     return integral
66
67 # intégration 3 : méthode de Simpson
68 def simpson_integration(f,x1,x2,n):
69     """
70     intégration de f(x) entre x1 et x2
71     par la méthode de Simpson
72
73     >>> fabs(simpson_integration(sin,0.,2*pi,100000)) < 1.e-5
74     True
75     >>> fabs(simpson_integration(sin,0.,pi,100000) - 2.) < 1.e-5
76     True
77     >>> fabs(simpson_integration(sin,0.,pi/2,100000) - 1) < 1.e-5
78     True
79     >>> fabs(simpson_integration(cos,0.,pi,100000)) < 1.e-5
80     True
81     >>> fabs(simpson_integration(cos,-pi/2,pi/2,100000) - 2) < 1.e-5
82     True
83     """
84     assert type(n) is int
85     assert type(x1) is float
86     assert type(x2) is float
87     assert x1 <= x2
88     assert n%2 == 0
89
90     integral = f(x1) + f(x2)
91     width = (x2 - x1)/n
92     for i in range(1,n,2):
93         integral = integral + 4*f(x1 + i*width)
94     for i in range(2,n,2):
95         integral = integral + 2*f(x1 + i*width)
96     integral = integral*width/3

```

```

97     return integral
98
99 #-----
100 if __name__ == "__main__":
101     import doctest
102     doctest.testmod()

```

TD 3.24 : TRACÉS DE COURBES PARAMÉTRÉES

```

1  #-*- coding: utf-8 -*-
2
3  from math import *
4  from turtle import *
5
6  #-----
7  def parametric_line(x0,y0,alpha,beta):
8      """
9      droite paramétrique
10     x = x0 + alpha*t, y = y0 + beta*t
11     """
12     return lambda t: x0 + alpha*t,\
13         lambda t: y0 + beta*t
14
15 #-----
16 def parametric_circle(x0,y0,r):
17     """
18     cercle paramétrique
19     x = x0 + r*cos(theta), y = y0 + r * sin(theta)
20     """
21     return lambda theta: x0 + r * cos(theta),\
22         lambda theta: y0 + r * sin(theta)
23
24 #-----
25 def parametric_ellipse(x0,y0,a,b):
26     """
27     ellipse paramétrique
28     x = x0 + a*cos(phi), y = y0 + b*sin(phi)
29     """
30     return lambda phi: x0 + a*cos(phi),\
31         lambda phi: y0 + b*sin(phi)
32
33 #-----
34 def parametric_hyperbola(x0,y0,a,b):
35     """
36     hyperbole paramétrique
37     x = x0 + a/cos(theta), y = y0 + b*tan(theta)
38     """
39     return lambda theta: x0 + a/cos(theta),\
40         lambda theta: y0 + b*tan(theta)
41
42 #-----
43 def parametric_cycloid(x0,y0,r):
44     """
45     cycloïde paramétrique
46     x = x0 + r*(phi-sin(phi)), y = y0 + r*(1-cos(phi))
47     """
48     return lambda phi: x0 + r*(phi-sin(phi)),\
49         lambda phi: y0 + r*(1-cos(phi))

```

```

50
51 #-----
52 def parametric_epicycloid(x0,y0,R,r):
53     """
54     épicycloïde paramétrique
55     x = x0 + (R+r)*cos(theta) - r*cos(theta*(R+r)/r),
56     x = y0 + (R+r)*sin(theta) - r*sin(theta*(R+r)/r)
57     """
58     return lambda theta: x0 + (R+r)*cos(theta) - r*cos(theta*(R+r)/r),\
59           lambda theta: y0 + (R+r)*sin(theta) - r*sin(theta*(R+r)/r)
60
61 #-----
62 def parametric_hypercycloid(x0,y0,R,r):
63     """
64     hypercycloïde paramétrique
65     x = x0 + (R-r)*cos(theta) + r*cos(theta*(R-r)/r),
66     x = y0 + (R-r)*sin(theta) + r*sin(theta*(R-r)/r)
67     """
68     return lambda theta: x0 + (R-r)*cos(theta) + r*cos(theta*(R-r)/r),\
69           lambda theta: y0 + (R-r)*sin(theta) + r*sin(theta*(R-r)/r)
70
71 #-----
72 def pascal_snail(x0,y0,a,b):
73     """
74     limaçon de Pascal
75     x = x0 + (a*cos(theta) + b)*cos(theta)
76     y = y0 + (a*cos(theta) + b)*sin(theta)
77     """
78     return lambda theta: x0 + (a*cos(theta) + b)*cos(theta),\
79           lambda theta: y0 + (a*cos(theta) + b)*sin(theta)
80
81 #-----
82 def logarithmic_spiral(x0,y0,k):
83     """
84     spirale logarithmique
85     x = x0 + k*exp(theta)*cos(theta)
86     y = y0 + k*exp(theta)*sin(theta)
87     """
88     return lambda theta: x0 + k*exp(theta)*cos(theta),\
89           lambda theta: y0 + k*exp(theta)*sin(theta)
90
91 #-----
92 def drawCurve(f,t1,t2,dt):
93     """
94     trace une courbe paramétrée pour t dans [t1,t2] par pas de dt
95     pour les fonctions x = fx(t) et y = fy(t) où f = (fx,fy)
96
97     >>> drawCurve(parametric_line(10,-10,2,3),-20.,20.,0.1)
98     >>> drawCurve(parametric_circle(10,-20,40),0.,2*pi,pi/100)
99     >>> drawCurve(parametric_ellipse(-30.,-10.,70,30),0.,2*pi,pi/100)
100    >>> drawCurve(parametric_hyperbola(-50.,0.,70,30),-1.,1.,0.1)
101    >>> drawCurve(parametric_cycloid(-150.,-100.,20.),0.,5*pi,pi/100)
102    >>> drawCurve(parametric_epicycloid(-100.,75.,40.,4.),0.,2*pi,pi/100)
103    >>> drawCurve(parametric_hypercycloid(100.,75.,40.,6.),0.,8*pi,pi/100)
104    >>> drawCurve(pascal_snail(-150.,0.,100.,80.),0.,2*pi,pi/100)
105    >>> drawCurve(logarithmic_spiral(100.,0.,0.1),0.,7.,pi/50)
106    """
107    assert type(t1) is float
108    assert type(t2) is float

```



```
109     assert type(dt) is float
110
111     (fx, fy) = f
112     values = []
113     t = t1 + dt
114     while t < t2:
115         values.append(t)
116         t = t + dt
117     up()
118     goto(fx(t1),fy(t1))
119     down()
120     for t in values:
121         goto(fx(t),fy(t))
122     return
123
124 #-----
125 if __name__ == "__main__":
126     import doctest
127     doctest.testmod()
```

Chapitre 4

Structures linéaires

TD 4.1 : DISTANCE DE 2 POINTS DE L'ESPACE

```
1 # -*- coding: utf-8 -*-
2
3 from math import sqrt
4
5 def distance(m1,m2):
6     assert type(m1) is tuple and len(m1) == 3
7     assert type(m2) is tuple and len(m2) == 3
8     (x1,y1,z1) = m1
9     (x2,y2,z2) = m2
10    d = (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + (z2-z1)*(z2-z1)
11    return sqrt(d)
```

TD 4.2 : OPÉRATIONS SUR LES N-UPLETS

```
1 # -*- coding: utf-8 -*-
2
3 # x in s
4 x = 3
5 s = (1,2,3,4)
6 if x in s:
7     print(x,'in',s,':',x in s)
8 else:
9     print(x,'not in',s,':',x not in s)
10
11 # x not in s
12 x = 0
13 s = (1,2,3,4)
14 if x in s:
15     print(x,'in',s,':',x in s)
16 else:
17     print(x,'not in',s,':',x not in s)
18
19 # s1 + s2
20 s1 = (1,2,3)
21 s2 = (4,5)
22 print(s1,'+',s2,'=',s1+s2,'!=',s2,'+',s1,'=',s2+s1,':',s1+s2 != s2+s1)
23
24 # s*n, n*s
25 n = 3
26 s = (1,2)
```

```

27 print(s, '*', n, '=', s*n, '=', n, '*', s, '=', n*s, ':', s*n == n*s)
28
29 # s[i]
30 s = (1,2,3)
31 for i in range(len(s)):
32     print(s, '[' , i, ']' =', s[i])
33
34 # s[i:j:step]
35 s = (0,1,2,3,4,5,6,7,8,9)
36 print('nombres pairs   :', s[0:10:2])
37 print('nombres impairs :', s[1:10:2])
38
39 # len(s)
40 s1 = (1,2,3)
41 s2 = (4,5)
42 print('len(', s1+s2, ') =', len(s1+s2), '=', len(s1), '+', len(s2), '= len(', s1, ') + len(', s2, ')')
43
44 # min(s)
45 s = (6,4,7,9,2,1,0,3,5,8)
46 print('min(', s, ') = ', min(s))
47
48 # max(s)
49 s = (6,4,7,9,2,1,0,3,5,8)
50 print('max(', s, ') = ', max(s))

```

TD 4.3 : PGCD ET PPCM DE 2 ENTIERS (2)

```

1 # -*- coding: utf-8 -*-
2
3 def pgcd_ppcm(a,b):
4     """
5     (pgcd, ppcm) = pgcd_ppcm(a,b)
6     plus grand commun diviseur et plus petit commun multiple
7     des 2 entiers a et b
8
9     >>> pgcd_ppcm(12,18)
10    (6, 36)
11    >>> pgcd_ppcm(21,15)
12    (3, 105)
13    >>> pgcd_ppcm(5,7)
14    (1, 35)
15    """
16    assert type(a) is int and a >= 0
17    assert type(b) is int and b >= 0
18
19    n1, n2 = a, b
20    while n2 != 0:
21        r = n1%n2
22        n1 = n2
23        n2 = r
24    pgcd = n1
25    if pgcd == 0: ppcm = 0
26    else       : ppcm = a*b//pgcd
27
28    return (pgcd,ppcm)
29
30 #-----
31 if __name__ == "__main__":

```

```

32     import doctest
33     doctest.testmod()

```

TD 4.4 : OPÉRATIONS SUR LES CHAÎNES

```

1  # -*- coding: utf-8 -*-
2
3  # s.capitalize()
4  s = 'bOnJoUr'
5  print('"' + s + '".capitalize() -> "' + s.capitalize() + '", sep='')
6
7  # s.count(sub[,start[,end]])
8  s = 'bonjour'
9  sub, start, end = 'o', 0, len(s)
10 print('"' + s + '".count("' + sub + '", ' + start + ', ' + end + ') -> ' + s.count(sub, start, end) + ', sep='')
11
12 # s.endswith(suffix[,start[,end]])
13 s = 'bonjour'
14 suffix, start, end = 'jour', 0, len(s)
15 print('"' + s + '".endswith("' + suffix + '", ' + start + ', ' + end + ') -> "' + s.endswith(suffix, start, end) + '", sep='')
16
17 # s.expandtabs([tabsize])
18 s = 'bonjour\t\tMonsieur'
19 tabsize = 1
20 print('"' + s + '".expandtabs(' + str(tabsize) + ') -> ' + s.expandtabs(tabsize) + ', sep='')
21
22 # s.find(sub[,start[,end]])/rfind(sub[,start[,end]])
23 s = 'bonjour'
24 sub, start, end = 'o', 0, len(s)
25 print('"' + s + '".find("' + sub + '", ' + start + ', ' + end + ') -> ' + s.find(sub, start, end) + ', sep='')
26 print('"' + s + '".rfind("' + sub + '", ' + start + ', ' + end + ') -> ' + s.rfind(sub, start, end) + ', sep='')
27
28 # s.index(sub[,start[,end]])/rindex(sub[,start[,end]])
29 s = 'bonjour'
30 sub, start, end = 'o', 0, len(s)
31 print('"' + s + '".index("' + sub + '", ' + start + ', ' + end + ') -> ' + s.index(sub, start, end) + ', sep='')
32 print('"' + s + '".rindex("' + sub + '", ' + start + ', ' + end + ') -> ' + s.rindex(sub, start, end) + ', sep='')
33
34 # s.isalnum()
35 s = 'bonjour2fois'
36 print('"' + s + '".isalnum() -> ' + s.isalnum() + ', sep='')
37
38 # s.isalpha()
39 s = 'bonjour2fois'
40 print('"' + s + '".isalpha() -> ' + s.isalpha() + ', sep='')
41
42 # s.isdigit()
43 s = '2175'
44 print('"' + s + '".isdigit() -> ' + s.isdigit() + ', sep='')
45
46 # s.isspace()
47 s = '\t \t\n'
48 print('"' + s + '".isspace() -> ' + s.isspace() + ', sep='')
49
50 # s.istitle()
51 s = 'Bonjour'
52 print('"' + s + '".istitle() -> ' + s.istitle() + ', sep='')
53

```

```

54 # s.islower()
55 s = 'bonjour'
56 print('','',s,".islower() -> ",s.islower(),sep='')
57
58 # s.isupper()
59 s = 'BONJOUR'
60 print('','',s,".isupper() -> ",s.isupper(),sep='')
61
62 # s.join(seq)
63 s = ''
64 seq = ['bon','j','our']
65 print('','',s,".join(' ',seq,'') -> ",s.join(seq),'','',sep='')
66
67 # s.ljust(width[,fillChar=' '])/rjust(width[,fillChar=' '])/center(width[,fillChar=' '])
68 s = 'bonjour'
69 width, fillChar = 14, '-'
70 print('','',s,".ljust(' ',width,' ',fillChar,'') -> ",s.ljust(width,fillChar),'','',sep='')
71 print('','',s,".rjust(' ',width,' ',fillChar,'') -> ",s.rjust(width,fillChar),'','',sep='')
72 print('','',s,".center(' ',width,' ',fillChar,'') -> ",s.center(width,fillChar),'','',sep='')
73
74 # s.lower()/title()/upper()
75 s = 'BonJouR'
76 print('','',s,".lower() -> ",s.lower(),'','',sep='')
77 print('','',s,".title() -> ",s.title(),'','',sep='')
78 print('','',s,".upper() -> ",s.upper(),'','',sep='')
79
80 # s.lstrip([chars])/rstrip([chars])/strip([chars])
81 s = 'bonjour'
82 chars = 'nob'
83 print('','',s,".lstrip(' ',chars,'') -> ",s.lstrip(chars),'','',sep='')
84 chars = 'ru'
85 print('','',s,".rstrip(' ',chars,'') -> ",s.rstrip(chars),'','',sep='')
86 chars = 'orb'
87 print('','',s,".strip(' ',chars,'') -> ",s.strip(chars),'','',sep='')
88
89 # s.partition(separ)/rpartition(separ)
90 s = 'bonjour'
91 separ = 'o'
92 print('','',s,".partition(' ',separ,'') -> ",s.partition(separ),sep='')
93 print('','',s,".rpartition(' ',separ,'') -> ",s.rpartition(separ),sep='')
94
95 # s.replace(old,new[,maxCount=-1])
96 s = 'bonjour'
97 old, new, maxCount = 'o', '0', 1
98 print('','',s,".replace(' ',old,' ',new,' ',maxCount,'') -> ",s.replace(old,new,maxCount),sep='')
99
100 # s.split([separ[,maxsplit]])/rsplit([separ[,maxsplit]])
101 s = 'bonjour'
102 separ, maxsplit = 'o', 1
103 print('','',s,".split(' ',separ,' ',maxsplit,'') -> ",s.split(separ,maxsplit),sep='')
104 print('','',s,".rsplit(' ',separ,' ',maxsplit,'') -> ",s.rsplit(separ,maxsplit),sep='')
105
106 # s.splitlines([keepends])
107 s = 'bonjour\nça va ?\nmerci\n'
108 keepends = True
109 print('','',s,".splitlines(' ',keepends,'') -> ",s.splitlines(keepends),sep='')
110 keepends = False
111 print('','',s,".splitlines(' ',keepends,'') -> ",s.splitlines(keepends),sep='')
112

```

```

113 # s.startswith(prefix[,start[,end]])
114 s = 'bonjour'
115 prefix, start, end = 'bo', 0, len(s)
116 print('"' + s + '".startswith('" + prefix + "'," + start + "," + end + ') -> ' + s.startswith(prefix, start, end), sep='')
117
118 # s.swapcase()
119 s = 'bonjour'
120 print('"' + s + '".swapcase() -> "' + s.swapcase() + '", sep='')
121
122 # s.zfill(width)
123 s = '-3.14'
124 width = 14
125 print('"' + s + '".zfill(' + width + ') -> "' + s.zfill(width) + '", sep='')

```

TD 4.5 : INVERSER UNE CHAÎNE

```

1 # -*- coding: utf-8 -*-
2
3 def inverser(s):
4     """
5     sinv = inverser(s)
6     chaîne dont les caractères sont dans l'ordre inverse de la chaîne s
7
8     >>> inverser('inverser')
9     'resrevni'
10    >>> inverser('kayak')
11    'kayak'
12    >>> inverser('')
13    ''
14    """
15    assert type(s) is str
16
17    sinv = ''
18    for c in s:
19        sinv = c + sinv
20
21    return sinv
22
23 #-----
24 if __name__ == "__main__":
25     import doctest
26     doctest.testmod()

```

TD 4.6 : CARACTÈRES, MOTS, LIGNES D'UNE CHAÎNE

```

1 # -*- coding: utf-8 -*-
2
3 def cml(s):
4     """
5     (nc, nm, nl) = cml(s)
6     nc est le nombre de caractères de la chaîne s,
7     nm le nombre de mots et nl le nombre de lignes
8
9     >>> s = "j'ai deux soeurs et 1 frère !"
10    >>> cml(s)
11    (29, 7, 1)

```

```

12     >>> s = "j'ai deux soeurs\\n et 1 frère !"
13     >>> cml(s)
14     (30, 7, 2)
15     >>> s = ''
16     >>> cml(s)
17     (0, 0, 0)
18     ""
19     assert type(s) is str
20
21     nc, nm, nl = 0, 0, 1
22     if s == '' : nl = 0
23
24     mot = False
25
26     for c in s:
27         if c.isalnum() :
28             if not mot:
29                 nm = nm + 1
30                 mot = True
31             elif c.isspace() :
32                 if c == '\n' : nl = nl + 1
33                 mot = False
34             else:
35                 mot = False
36                 nc = nc + 1
37     assert nc == len(s) # vérification
38
39     return nc, nm, nl
40
41 #-----
42 if __name__ == "__main__":
43     import doctest
44     doctest.testmod()

```

TD 4.7 : OPÉRATIONS SUR LES LISTES (1)

```

1  # -*- coding: utf-8 -*-
2
3  # s[i] = x
4  l = [1,2,3,4,5,6]
5  n, x = 2, 0
6  print(l,['',n,''] = ',x,' : ',l,' -> ',sep='',end='')
7  l[n] = x
8  print(l)
9
10 # s[i:j] = t
11 l = [1,2,3,4,5,6]
12 i, j = 1, 3
13 t = [-4, -5]
14 print(l,['',i,':',j,''] = ',t,' : ',l,' -> ',sep='',end='')
15 l[i:j] = t
16 print(l)
17
18 # del s[i:j]          same as s[i:j] = []
19 l = [1,2,3,4,5,6]
20 i, j = 1, 3
21 print('del ',l,['',i,':',j,''] : ',l,' -> ',sep='',end='')
22 del l[i:j]

```



```

23 print(l)
24
25 # s[i:j:k] = t          the elements of s[i:j:k] are replaced by those of t      (1)
26 l = [1,2,3,4,5,6]
27 i, j, k = 1, 6, 2
28 t = [-4,-5,-6]
29 print(l, '[' ,i, ':' ,j, ':' ,k, ']' = ',t,' : ',l,' -> ',sep='',end='')
30 l[i:j:k] = t
31 print(l)
32
33 # del s[i:j:k]          removes the elements of s[i:j:k] from the list
34 l = [1,2,3,4,5,6]
35 i, j, k = 1, 6, 2
36 print('del ',l,[' ,i, ':' ,j, ':' ,k, ']' : ',l,' -> ',sep='',end='')
37 del l[i:j:k]
38 print(l)
39
40 # s.append(x)
41 l = [1,2,3,4,5,6]
42 x = 0
43 print(l, '.append(',x,') : ',l,' -> ',sep='',end='')
44 l.append(x)
45 print(l)
46
47 # s.extend(t)           extends s with the contents of t (same as s[len(s):len(s)] = t)
48 l = [1,2,3]
49 t = [7,8,9]
50 print(l, '.extend(',t,') : ',l,' -> ',sep='',end='')
51 l.extend(t)
52 print(l)
53
54 # s.insert(i, x)
55 l = [4,5,6]
56 i, x = 1, -4
57 print(l, '.insert(',i, ',' ,x,') -> ',sep='',end='')
58 l.insert(i,x)
59 print(l)
60
61 # s.pop([i])
62 l = [2,3,4,5,6]
63 i = 1
64 print(l, '.pop(',i,') : ',l,' -> ',sep='',end='')
65 l.pop(i)
66 print(l)
67
68 # s.remove(x)
69 l = [2,3,4,5,4,3]
70 x = 4
71 print(l, '.remove(',x,') : ',l,' -> ',sep='',end='')
72 l.remove(x)
73 print(l)
74
75 # s.reverse()
76 l = [1,2,3]
77 print(l, '.reverse() : ',l,' -> ',sep='',end='')
78 l.reverse()
79 print(l)
80
81 # s.sort()

```

```

82 l = [8,5,9,2,0,1,5]
83 print(l, '.sort() : ', l, ' -> ', sep='', end='')
84 l.sort()
85 print(l)

```

TD 4.8 : OPÉRATIONS SUR LES LISTES (2)

```

1  # -*- coding: utf-8 -*-
2
3  # del s[i:j] et s[i:j] = []
4  s = [1,2,3,4,5,6]
5  l = [1,2,3,4,5,6]
6  i, j = 1, 3
7  del s[i:j]
8  l[i:j] = []
9  print('del s[i:j] et s[i:j] = [] : ', l == s)
10
11 # s.append(x) et s[len(s):len(s)] = [x]
12 s = [1,2,3,4,5,6]
13 l = [1,2,3,4,5,6]
14 x = 0
15 s.append(x)
16 l[len(l):len(l)] = [x]
17 print('s.append(x) et s[len(s):len(s)] = [x] : ', l == s)
18
19 # s.extend(x) et s[len(s):len(s)] = x
20 s = [1,2,3,4,5,6]
21 l = [1,2,3,4,5,6]
22 x = [-3,-4]
23 s.extend(x)
24 l[len(l):len(l)] = x
25 print('s.extend(x) et s[len(s):len(s)] = x : ', l == s)
26
27 # s.insert(i,x) et s[i:i] = [x]
28 s = [1,2,3,4,5,6]
29 l = [1,2,3,4,5,6]
30 x = -3
31 i = 1
32 s.insert(i,x)
33 l[i:i] = [x]
34 print('s.insert(i,x) et s[i:i] = [x] : ', l == s)
35
36 # s.remove(x) et del s[s.index(x)]
37 s = [1,2,3,4,5,6]
38 l = [1,2,3,4,5,6]
39 x = 3
40 s.remove(x)
41 del l[l.index(x)]
42 print('s.remove(x) et del s[s.index(x)] : ', l == s)
43
44 # s.pop(i) et x = s[i]; del s[i]
45 s = [4,5,6,7]
46 l = [4,5,6,7]
47 i = 2
48 s.pop(i)
49 x = l[i]; del l[i]
50 print('s.pop(i) et x = s[i]; del s[i] : ', l == s)

```

TD 4.9 : SÉLECTION D'ÉLÉMENTS

```

1 # -*- coding: utf-8 -*-
2
3 from math import *
4
5 p1 = lambda e : e%2 == 0
6 p2 = lambda e : sqrt(e) == int(sqrt(e))
7
8 #-----
9 def selection(s,p):
10     """
11     t = selection(s,p)
12     liste des éléments de la liste s qui vérifient
13     la condition p (p(s[i]) == True)
14
15     >>> selection([0,1,2,3,4,5,6],p1)
16     [0, 2, 4, 6]
17     >>> selection([2,4,6,9],p2)
18     [4, 9]
19     """
20     assert type(s) is list
21
22     t = []
23     for e in s:
24         if p(e) == True :
25             t.append(e)
26
27     return t
28
29 #-----
30 if __name__ == "__main__":
31     import doctest
32     doctest.testmod()

```

TD 4.10 : OPÉRATIONS SUR LES PILES

```

1 # -*- coding: utf-8 -*-
2
3 # pile (structure LIFO : last in first out) :
4 # on empile et dépile à la fin de la liste
5 # qui stocke les éléments de la pile
6
7 #-----
8 def emptyStack(p):
9     """
10     ok = emptyStack(p)
11     True si p est une liste vide, False sinon
12
13     >>> emptyStack([])
14     True
15     >>> emptyStack([[]])
16     False
17     >>> emptyStack([1,2,3])
18     False

```

```

19     """
20     assert type(p) is list
21
22     return (p == [])
23
24 #-----
25 def topStack(p):
26     """
27     x = topStack(p)
28     sommet d'une pile p non vide (not emptyStack(p))
29
30     >>> topStack([[]])
31     []
32     >>> topStack([1,2,3])
33     3
34     """
35     assert not emptyStack(p)
36
37     return p[len(p)-1]
38
39 #-----
40 def pushStack(p,x):
41     """
42     pushStack(p,x) empile x sur le sommet de la pile p
43
44     >>> p = []; pushStack(p,2); print(p)
45     [2]
46     >>> p = [1,2,3]; pushStack(p,2); print(p)
47     [1, 2, 3, 2]
48     >>> p = [[]]; pushStack(p,2); print(p)
49     [[], 2]
50     """
51     assert type(p) is list
52
53     p.append(x)
54
55     return
56
57 #-----
58 def popStack(p):
59     """
60     x = popStack(p) dépile le sommet x d'une pile p non vide
61
62     >>> p = [1,2,3]; popStack(p); print(p)
63     3
64     [1, 2]
65     >>> p = [[]]; popStack(p); print(p)
66     []
67     []
68     """
69     assert not emptyStack(p)
70
71     x = topStack(p)
72     del p[len(p)-1]
73
74     return x
75
76 #-----
77 if __name__ == "__main__":

```

```

78     import doctest
79     doctest.testmod()

```

TD 4.11 : OPÉRATIONS SUR LES FILES

```

1  # -*- coding: utf-8 -*-
2
3  # file (structure FIFO : first in first out) :
4  # on enfile au début et défile à la fin de la liste
5  # qui stocke les éléments de la file
6
7  #-----
8  def emptyQueue(p):
9      """
10     ok = emptyQueue(p)
11     True si p est une liste vide, False sinon
12
13     >>> emptyQueue([])
14     True
15     >>> emptyQueue([[]])
16     False
17     >>> emptyQueue([1,2,3])
18     False
19     """
20     assert type(p) is list
21
22     return (p == [])
23
24 #-----
25 def topQueue(p):
26     """
27     x = topQueue(p)
28     tête d'une file p non vide (not emptyQueue(p))
29
30     >>> topQueue([[]])
31     []
32     >>> topQueue([1,2,3])
33     3
34     """
35     assert not emptyQueue(p)
36
37     return p[len(p)-1]
38
39 #-----
40 def pushQueue(p,x):
41     """
42     pushQueue(p,x) enfile x dans la file p
43
44     >>> p = []; pushQueue(p,2); print(p)
45     [2]
46     >>> p = [1,2,3]; pushQueue(p,2); print(p)
47     [2, 1, 2, 3]
48     >>> p = [[]]; pushQueue(p,2); print(p)
49     [2, []]
50     """
51     assert type(p) is list
52
53     p.insert(0,x)

```

```

54
55     return
56
57 #-----
58 def popQueue(p):
59     """
60     x = popQueue(p) défile le sommet x d'une file p non vide
61
62     >>> p = [1,2,3]; popQueue(p); print(p)
63     3
64     [1, 2]
65     >>> p = [[]]; popQueue(p); print(p)
66     []
67     []
68     """
69     assert not emptyQueue(p)
70
71     x = topQueue(p)
72     del p[len(p)-1]
73
74     return x
75
76 #-----
77 if __name__ == "__main__":
78     import doctest
79     doctest.testmod()

```

TD 4.12 : PRODUIT DE MATRICES

```

1 # -*- coding: utf-8 -*-
2
3
4 #-----
5 def matrice(a):
6     """
7     ok = matrice(a)
8     teste si a est une matrice
9
10    >>> matrice([[1,2],[3,4],[5,6]])
11    True
12    >>> matrice([[1,2],[3,4],[5,6,7]])
13    False
14    >>> matrice([])
15    False
16    >>> matrice([],[],[])
17    False
18    """
19    ok = False
20    if type(a) is list and len(a) != 0 and\
21        type(a[0]) is list and len(a[0]) != 0 :
22        ok = True
23        arow = len(a)
24        acol = len(a[0])
25
26        i = 1
27        while i < len(a) and ok :
28            if type(a[i]) is not list or len(a[i]) != len(a[0]) :
29                ok = False

```

```

30         else :
31             i = i + 1
32
33     return ok
34
35 #-----
36 def nrow(a) :
37     """
38     n = nrow(a)
39     nombre de lignes de la matrice a
40
41     >>> nrow([[1,2],[3,4],[5,6]])
42     3
43     >>> nrow([[1,2,3,4]])
44     1
45     """
46     assert matrice(a)
47
48     return len(a)
49
50 #-----
51 def ncol(a) :
52     """
53     n = ncol(a)
54     nombre de colonnes de la matrice a
55
56     >>> ncol([[1,2],[3,4],[5,6]])
57     2
58     >>> ncol([[1,2,3,4]])
59     4
60     """
61     assert matrice(a)
62
63     return len(a[0])
64
65 #-----
66 def multMatrice(a,b):
67     """
68     c = multMatrice(a,b)
69     produit matriciel des matrices a et b
70
71     >>> a = [[1,1,1],[2,4,8],[3,9,27]]
72     >>> b = [[-1],[1],[1]]
73     >>> multMatrice(a,b)
74     [[1], [10], [33]]
75     >>> a = [[2,-1,2],[1,10,-3],[-1,2,1]]
76     >>> b = [[2],[0],[-1]]
77     >>> multMatrice(a,b)
78     [[2], [5], [-3]]
79     >>> a = [[1,1],[2,2],[3,3]]
80     >>> b = [[-1,-1,-1],[-2,-2,-2]]
81     >>> multMatrice(a,b)
82     [[-3, -3, -3], [-6, -6, -6], [-9, -9, -9]]
83     >>> a = [[2]]
84     >>> b = [[1,2,3,4,5]]
85     >>> multMatrice(a,b)
86     [[2, 4, 6, 8, 10]]
87     """
88     assert matrice(a) and matrice(b)

```

```

89     assert ncol(a) == nrow(b)
90
91     arow, acol = nrow(a), ncol(a)
92     brow, bcol = nrow(b), ncol(b)
93
94     c = []
95     for i in range(arow) :
96         c.append([])
97         for j in range(bcol) :
98             c[i].append(0)
99
100    for i in range(arow) :
101        for j in range(bcol) :
102            for k in range(acol) :
103                c[i][j] = c[i][j] + a[i][k]*b[k][j]
104
105    return c
106
107 #-----
108 if __name__ == "__main__":
109     import doctest
110     doctest.testmod()

```

TD 4.13 : ANNUAIRE TÉLÉPHONIQUE

```

1  # -*- coding: utf-8 -*-
2
3  # recherche dans un annuaire
4
5  #-----
6  def annuaire(a) :
7      """
8      ok = annuaire(a)
9      True si a est un annuaire : une liste de paires (nom,numéro)
10
11      >>> annuaire([])
12      True
13      >>> annuaire([('jean','0607080910'),('paul','0298000102')])
14      True
15      >>> annuaire([1,2,4])
16      False
17      """
18      ok = False
19      if type(a) is list :
20          i = 0
21          ok = True
22          while i < len(a) and ok :
23              if type(a[i]) is not tuple or len(a[i]) != 2 :
24                  ok = False
25              else :
26                  i = i + 1
27
28      return ok
29
30 #-----
31 def numero(nom,a) :
32     """
33     num = numero(nom,a)

```



```

34     numero correspondant au nom dans l'annuaire a
35
36     >>> annuaire = [('jean','0607080910'),('paul','0298000102')]
37     >>> numero('paul',annuaire)
38     '0298000102'
39     >>> numero('pierre',annuaire)
40     ''
41     """
42     assert annuaire(a)
43
44     num, i = '', 0
45     while i < len(a) and num == '' :
46         if a[i][0] == nom :
47             num = a[i][1]
48         else:
49             i = i + 1
50
51     return num
52
53 #-----
54 def nom(numero,a) :
55     """
56     n = nom(numero,a)
57     nom correspondant au numero dans l'annuaire a
58
59     >>> annuaire = [('jean','0607080910'),('paul','0298000102')]
60     >>> nom('0298000102',annuaire)
61     'paul'
62     >>> nom('0298000000',annuaire)
63     ''
64     >>> nom(numero('paul',annuaire),annuaire) == 'paul'
65     True
66     """
67     assert annuaire(a)
68
69     n, i = '', 0
70     while i < len(a) and n == '' :
71         if a[i][1] == numero :
72             n = a[i][0]
73         else :
74             i = i + 1
75
76     return n
77
78 #-----
79 if __name__ == "__main__":
80     import doctest
81     doctest.testmod()

```

TD 4.14 : RECHERCHE DICHOTOMIQUE

```

1 # -*- coding: utf-8 -*-
2
3 # la recherche dichotomique n'assure pas automatiquement
4 # de trouver la première occurrence de la valeur recherchée
5
6 #-----
7 def dichotomie(t,x,gauche,droite) :

```

```

8      """
9      ok,m = dichotomie(t,x,gauche,droite)
10     recherche dichotomique de x dans une séquence t triée
11     entre les indices gauche et droite
12     ok == True si x a été trouvée à l'indice m,
13     False sinon
14
15     >>> t = [1,3,5,6,6,6,6,9]
16     >>> dichotomie(t,6,0,len(t)-1)
17     (True, 3)
18     >>> t = [1,3,5,6,6,9]
19     >>> dichotomie(t,6,0,len(t)-1)
20     (True, 4)
21     """
22     assert type(t) is list
23     assert 0 <= gauche <= droite < len(t)
24
25     ok, m = False, (gauche + droite)//2
26     if gauche > droite :
27         ok = False
28     else :
29         if t[m] == x :
30             ok = True
31         elif t[m] > x :
32             ok,m = dichotomie(t,x,gauche,m-1)
33         else :
34             ok,m = dichotomie(t,x,m+1,droite)
35
36     return ok,m
37
38 #-----
39 def dichotomie1(t,x,gauche,droite) :
40     """
41     ok,m = dichotomie1(t,x,gauche,droite)
42     recherche dichotomique de la première occurrence de x
43     dans une séquence t triée
44     entre les indices gauche et droite
45     ok == True si x a été trouvée à l'indice m,
46     False sinon
47
48     >>> t = [1,3,5,6,6,6,6,9]
49     >>> dichotomie1(t,6,0,len(t)-1)
50     (True, 3)
51     >>> t = [1,3,5,6,6,9]
52     >>> dichotomie1(t,6,0,len(t)-1)
53     (True, 3)
54     >>> dichotomie(t,6,0,len(t)-1) == dichotomie1(t,6,0,len(t)-1)
55     False
56     """
57     assert type(t) is list
58     assert 0 <= gauche <= droite < len(t)
59
60     ok, m = False, (gauche + droite)//2
61     if gauche > droite :
62         ok = False
63     else :
64         if t[m] == x :
65             ok = True
66             m1 = m

```

```

67         # recherche de la 1ère occurrence
68         while t[m1] == x and m1 >= 0 :
69             m1 = m1 - 1
70             if t[m1] == x : m = m1;
71     elif t[m] > x :
72         ok,m = dichotomie1(t,x,gauche,m-1)
73     else :
74         ok,m = dichotomie1(t,x,m+1,droite)
75
76     return ok,m
77
78 #-----
79 if __name__ == "__main__":
80     import doctest
81     doctest.testmod()

```

TD 4.15 : LISTE ORDONNÉE

```

1  # -*- coding: utf-8 -*-
2
3  # enOrdre 1 : version récursive
4  def enOrdreRecuratif(t,debut,fin):
5      """
6      ok = enOrdreRecuratif(t)
7      teste si la liste t est ordonnée par ordre croissant
8
9      >>> t = [0,1,2,3,4]
10     >>> enOrdreRecuratif(t,0,len(t)-1)
11     True
12     >>> t = [4,1,2,3,0]
13     >>> enOrdreRecuratif(t,0,len(t)-1)
14     False
15     >>> enOrdreRecuratif(t,1,3)
16     True
17     """
18     assert type(t) is list
19     assert 0 <= debut <= fin < len(t)
20
21     ok = False
22     if debut == fin :
23         ok = True
24     else:
25         if t[debut] <= t[debut+1] :
26             ok = enOrdreRecuratif(t,debut+1,fin)
27     return ok
28
29  # enOrdre 2 : version itérative
30  def enOrdreIteratif(t,debut,fin):
31      """
32      ok = enOrdreIteratif(t)
33      teste si la liste t est ordonnée par ordre croissant
34
35      >>> t = [0,1,2,3,4]
36      >>> enOrdreIteratif(t,0,len(t)-1)
37      True
38      >>> t = [4,1,2,3,0]
39      >>> enOrdreIteratif(t,0,len(t)-1)
40      False

```

```

41     >>> enOrdreIteratif(t,1,3)
42     True
43     """
44     assert type(t) is list
45     assert 0 <= debut <= fin < len(t)
46
47     ok = True
48     while not debut == fin and ok:
49         if t[debut] <= t[debut+1] :
50             debut = debut + 1
51         else :
52             ok = False
53
54     return ok
55
56 #-----
57 if __name__ == "__main__":
58     import doctest
59     doctest.testmod()

```

TD 4.16 : TRI D'UN ANNUAIRE TÉLÉPHONIQUE

```

1  # -*- coding: utf-8 -*-
2
3  # tri d'un annuaire par ordre alphabétique
4
5  #-----
6  def annuaire(a) :
7      """
8      ok = annuaire(a)
9      True si a est une liste de paires (nom,numéro)
10
11      >>> annuaire([])
12      True
13      >>> annuaire([('jean','0607080910'),('paul','0298000102')])
14      True
15      >>> annuaire([1,2,4])
16      False
17      """
18      ok = False
19      if type(a) is list :
20          i = 0
21          ok = True
22          while i < len(a) and ok :
23              if type(a[i]) is not tuple or len(a[i]) != 2 :
24                  ok = False
25              else :
26                  i = i + 1
27
28      return ok
29
30 #-----
31 def minimum(a,debut,fin) :
32     """
33     mini = minimum(a,debut,fin)
34     indice du plus petit nom par ordre alphabétique
35     de l'annuaire a entre les indices debut et fin
36

```

```

37     >>> annuaire = [('jean','0607080910'),('paul','0298000102'),('albert','0300121314')]
38     >>> minimum(annuaire,0,2)
39     2
40     >>> minimum(annuaire,0,1)
41     0
42     """
43     assert annuaire(a)
44     assert 0 <= debut <= fin < len(a)
45
46     mini = debut
47     for j in range(debut+1,fin+1) :
48         if a[j][0] < a[mini][0] :
49             mini = j
50
51     return mini
52
53 #-----
54 def triAnnuaire(a,debut,fin) :
55     """
56     triAnnuaire(a,debut,fin)
57     tri l'annuaire a par ordre alphabétique des noms entre les indices debut et fin
58
59     >>> annuaire = [('jean','0607080910'),('paul','0298000102'),('albert','0300121314')]
60     >>> triAnnuaire(annuaire,0,2)
61     >>> annuaire
62     [('albert', '0300121314'), ('jean', '0607080910'), ('paul', '0298000102')]
63     >>> annuaire = [('jean','0607080910'),('paul','0298000102'),('albert','0300121314')]
64     >>> triAnnuaire(annuaire,0,1)
65     >>> annuaire
66     [('jean', '0607080910'), ('paul', '0298000102'), ('albert', '0300121314')]
67     """
68     assert annuaire(a)
69     assert 0 <= debut <= fin < len(a)
70
71     i = debut
72     while i < fin+1 :
73         mini = minimum(a,i,fin)
74         a[i], a[mini] = a[mini], a[i]
75         i = i + 1
76
77     return
78
79 #-----
80 if __name__ == "__main__":
81     import doctest
82     doctest.testmod()

```

TD 4.17 : COMPLEXITÉ DU TRI PAR SÉLECTION

```

1 # -*- coding: utf-8 -*-
2
3 #-----
4 def triSelection(t,debut,fin):
5     assert type(t) is list
6     assert 0 <= debut <= fin < len(t)
7     while debut < fin:
8         mini = minimum(t,debut,fin)
9         t[debut],t[mini] = t[mini],t[debut]

```

```

10     debut = debut + 1
11     return
12
13 #-----
14 def minimum(t,debut,fin):
15     assert type(t) is list
16     assert 0 <= debut <= fin < len(t)
17     mini = debut
18     for j in range(debut+1,fin+1):
19         if t[j] < t[mini]: mini = j
20         print(j,end=' ')
21     print()
22     return mini
23
24
25 #-----
26 # tri par sélection 1 : échanges de valeurs en O(n)
27 solution1 = ""1.
28 Les échanges de valeurs interviennent dans l'instruction
29     t[debut],t[mini] = t[mini],t[debut]
30 qui est présente dans le corps de la boucle
31     while debut < fin:
32         mini = minimum(t,debut,fin)
33         t[debut],t[mini] = t[mini],t[debut]
34         debut = debut + 1
35 Or, on passe (fin-debut) fois dans cette boucle, soit
36 l'ordre de grandeur du nombre d'éléments à trier : O(n).
37 ""
38 print(solution1)
39
40 #-----
41 # tri par sélection 2 : comparaisons entre éléments en O(n^2)
42 solution2 = ""2.
43 Les comparaisons entre éléments (t[j] < t[mini]) interviennent
44 dans le corps de la boucle de la fonction minimum
45     for j in range(debut+1,fin+1):
46         if t[j] < t[mini]: mini = j
47 On passe (fin-debut) fois dans cette boucle,
48 soit l'ordre de grandeur du nombre d'éléments à comparer à chaque
49 appel de la fonction minimum. La fonction minimum est elle-même
50 appelée dans le corps de la boucle
51     while debut < fin:
52         mini = minimum(t,debut,fin)
53         t[debut],t[mini] = t[mini],t[debut]
54         debut = debut + 1
55 Au premier passage, on effectue n comparaisons, (n-1) au 2ème passage, (n-2)
56 au troisième passage et ainsi de suite, soit en tout s comparaisons avec
57 s = n + (n-1) + (n-2) + ... + 1.
58 s est donc la somme des n premiers nombres entiers, à savoir
59 s = n*(n+1)/2, donc de l'ordre de n^2 : O(n^2).
60 ""
61 print(solution2)

```

TD 4.18 : TRI PAR INSERTION

```

1 # -*- coding: utf-8 -*-
2
3 def triInsertion(t,debut,fin):

```

```

4     assert type(t) is list
5     assert 0 <= debut <= fin < len(t)
6     for k in range(debut+1,fin+1):
7         i = k - 1
8         x = t[k]
9         while i >= debut and t[i] > x:
10             t[i+1] = t[i]
11             i = i - 1
12             print(i,k,x,t) # affichage
13         t[i+1] = x
14     return
15
16 triInsertion([9,8,7,6,5,4],1,4)

```

TD 4.19 : COMPARAISON D'ALGORITHMES (1)

```

1 # -*- coding: utf-8 -*-
2
3
4 #-----
5 def triRapideIteratif(t,debut,fin):
6     assert type(t) is list
7     assert 0 <= debut
8     assert fin <= len(t)
9     pile = []
10    while True:
11        while debut < fin:
12            pivot = t[debut]
13            place = partition(t,debut,fin,pivot)
14            empiler(pile,(debut,fin,place))
15            fin = place - 1
16        if not len(pile) == 0:
17            (debut,fin,place) = depiler(pile)
18            debut = place + 1
19        else: return
20    return
21
22 #-----
23 def triRapideRekursif(t,debut,fin):
24     assert type(t) is list
25     assert 0 <= debut
26     assert fin <= len(t)
27     if debut < fin:
28         pivot = t[debut]
29         place = partition(t,debut,fin,pivot)
30         triRapideRekursif(t,debut,place-1)
31         triRapideRekursif(t,place+1,fin)
32    return
33
34 #-----
35 def partition(t,debut,fin,pivot):
36     p,inf,sup = debut,debut,fin;
37     while p <= sup:
38         if t[p] == pivot:
39             p = p + 1
40         elif t[p] < pivot:
41             t[inf],t[p] = t[p],t[inf]
42             inf = inf + 1

```

```

43         p = p + 1
44     else:
45         t[sup],t[p] = t[p],t[sup]
46         sup = sup - 1
47     if p > 0: p = p - 1
48     return p
49
50 #-----
51 def empiler(p,x):
52     assert type(p) is list
53     p.append(x)
54     return
55
56 #-----
57 def depiler(p):
58     assert type(p) is list
59     assert len(p) > 0
60     x = p[len(p)-1]
61     del p[len(p)-1]
62     return x
63
64 #-----
65 from random import randint
66 from time import time
67
68 for n in [10,100,1000,10000,100000,1000000,10000000] :
69     s1, s2 = [], []
70     for k in range(n) :
71         x = randint(0,10*n)
72         s1.append(x)
73         s2.append(x)
74     t1 = time()
75     triRapideRecuratif(s1,0,len(s1)-1)
76     t2 = time()
77     print('récuratif : ',n,t2-t1)
78     t1 = time()
79     triRapideIteratif(s2,0,len(s2)-1)
80     t2 = time()
81     print('itératif : ',n,t2-t1)
82
83 """Exemples de temps obtenus :
84 récuratif : 10 0.00011992454528808594
85 itératif : 10 4.887580871582031e-05
86 récuratif : 100 0.0007030963897705078
87 itératif : 100 0.0012509822845458984
88 récuratif : 1000 0.010165929794311523
89 itératif : 1000 0.009513139724731445
90 récuratif : 10000 0.12292003631591797
91 itératif : 10000 0.13573694229125977
92 récuratif : 100000 1.550318956375122
93 itératif : 100000 1.585007905960083
94 récuratif : 1000000 18.29716205596924
95 itératif : 1000000 18.76704502105713
96 récuratif : 10000000 233.44398999214172
97 itératif : 10000000 243.83070611953735
98 """

```

TD 4.20 : QCM (4)

```

1 # -*- coding: utf-8 -*-
2
3 """
4 Les bonnes réponses sont extraites directement des notes de cours :
5 1c, 2d, 3d, 4c, 5b, 6d, 7c, 8a, 9c, 10b
6 """

```

TD 4.21 : GÉNÉRATION DE SÉQUENCES

```

1 # -*- coding: utf-8 -*-
2
3 from random import randint
4
5 # Génération de listes d'entiers.
6 def liste(n):
7     assert type(n) is int and n >= 0
8     t = []
9     for i in range(n):
10         t.append(randint(0,n))
11     return t
12
13 # Génération de n-uplets d'entiers.
14 def nuplet(n):
15     assert type(n) is int and n >= 0
16     t = ()
17     for i in range(n):
18         t = t + (randint(0,n),)
19     return t
20
21 # Génération de chaînes de caractères.
22 def chaine(n):
23     assert type(n) is int and n >= 0
24     t = ''
25     for i in range(n):
26         t = t + chr(randint(32,127))
27     return t
28
29 # Exemples
30 for n in [0,5,10] :
31     print(n,liste(n),nuplet(n),'',chaine(n),'')

```

TD 4.22 : APPLICATION D'UNE FONCTION À TOUS LES ÉLÉMENTS D'UNE LISTE

```

1 # -*- coding: utf-8 -*-
2
3 def application(f,t):
4     """
5     t = application(f,t)
6     applique la fonction f à chaque élément de la liste t
7
8     >>> s = [-1,0,-4,3,5,-7]
9     >>> application(abs,s)
10    [1, 0, 4, 3, 5, 7]
11    >>> s = [0,1,2,3]

```

```

12     >>> f = lambda x: x+2
13     >>> application(f,s)
14     [2, 3, 4, 5]
15     """
16     assert type(t) is list
17
18     for i in range(len(t)):
19         t[i] = f(t[i])
20
21     return t
22
23 #-----
24 if __name__ == "__main__":
25     import doctest
26     doctest.testmod()

```

TD 4.23 : QUE FAIT CETTE PROCÉDURE ?

```

1 # -*- coding: utf-8 -*-
2
3 def f(t,debut,fin):
4     m = (debut + fin)//2
5     while m > 0:
6         for i in range(m,fin+1):
7             j = i - m
8             while j >= debut:
9                 print(m,i,j,t)
10                 if t[j] > t[j+m]:
11                     t[j],t[j+m] = t[j+m],t[j]
12                     j = j - m
13                 else: j = debut-1
14             m = m//2
15     return t
16
17 t = [4,2,1,2,3,5]
18 print('avant :',t)
19 print('après :',f(t,0,5))

```

TD 4.24 : CODES ASCII ET CHAÎNES DE CARACTÈRES

```

1 # -*- coding: utf-8 -*-
2
3 # chaîne -> code ASCII
4 def codeASCII(s):
5     """
6     >>> codeASCII('bon')
7     [98, 111, 110]
8     """
9     assert type(s) is str
10
11     code = []
12     for i in range(len(s)):
13         code.append(ord(s[i]))
14     return code
15
16 # Codes ASCII -> chaîne

```

```

17 def decodeASCII(code):
18     """
19     >>> decodeASCII([98, 111, 110])
20     'bon'
21     """
22     assert type(code) is list
23
24     s = ''
25     for i in range(len(code)):
26         s = s + chr(code[i])
27     return s
28
29 #-----
30 if __name__ == "__main__":
31     import doctest
32     doctest.testmod()

```

TD 4.25 : OPÉRATIONS SUR LES MATRICES

```

1 # -*- coding: utf-8 -*-
2
3 # Matrice
4 def matrice(t):
5     """
6     ok = matrice(t)
7     True si t est une matrice, False sinon
8
9     >>> matrice(5)
10    False
11    >>> matrice([])
12    False
13    >>> matrice([5])
14    False
15    >>> matrice([[5]])
16    True
17    >>> matrice([[5,6],[7]])
18    False
19    >>> matrice([[5,6,7],[8,9]])
20    False
21    >>> matrice([[5,6,7],[8,9,3]])
22    True
23    """
24    ok = True
25    if type(t) is not list or t == []:
26        ok = False
27    elif type(t[0]) is not list:
28        ok = False
29    else:
30        i, n, m = 1, len(t), len(t[0])
31        while i < n and ok == True:
32            if type(t[i]) is not list:
33                ok = False
34            elif len(t[i]) != m:
35                ok = False
36            i = i + 1
37
38    return ok
39

```

```

40 # Matrice carrée
41 def matriceCarree(t):
42     """
43     ok = matriceCarree(t)
44     True si t est une matrice carrée, False sinon
45
46     >>> matriceCarree([[4,5,6],[7,8,9]])
47     False
48     >>> matriceCarree([[5,6],[8,9]])
49     True
50     >>> matriceCarree([])
51     False
52     """
53     assert matrice(t)
54
55     if len(t) > 0 and len(t[0]) == len(t):
56         ok = True
57     else: ok = False
58
59     return ok
60
61 # Matrice symétrique
62 def matriceSymetrique(t):
63     """
64     ok = matriceSymetrique(t)
65     True si t est une matrice carrée symétrique, False sinon
66
67     >>> matriceSymetrique([[5,6],[8,9]])
68     False
69     >>> matriceSymetrique([[5,6],[6,9]])
70     True
71     >>> matriceSymetrique([[5,6,7],[6,8,9],[7,9,3]])
72     True
73     """
74     assert matriceCarree(t)
75
76     ok,i = True,0
77     while i < len(t) and ok == True:
78         j = i + 1
79         while j < len(t[0]) and ok == True:
80             if t[i][j] != t[j][i]:
81                 ok = False
82             else: j = j + 1
83         i = i + 1
84
85     return ok
86
87 # Matrice diagonale
88 def matriceDiagonale(t):
89     """
90     ok = matriceDiagonale(t)
91     True si t est une matrice carrée diagonale, False sinon
92
93     >>> matriceDiagonale([[5,6],[8,9]])
94     False
95     >>> matriceDiagonale([[5,0],[0,9]])
96     True
97     >>> matriceDiagonale([[5,0,0],[0,0,0],[0,0,3]])
98     True

```

```

99     """
100     assert matriceCarree(t)
101
102     ok,i = True,0
103     while i < len(t) and ok == True:
104         j = 0
105         while j < len(t[0]) and ok == True:
106             if i != j and t[i][j] != 0:
107                 ok = False
108             else: j = j + 1
109         i = i + 1
110
111     return ok
112
113 # Multiplication d'une matrice par un scalaire
114 def multiplicationScalaire(t,x):
115     """
116     multiplicationScalaire(t,x)
117     multiplie la matrice t par le nombre x
118
119     >>> multiplicationScalaire([[5,6],[2,7]],3)
120     [[15, 18], [6, 21]]
121     """
122     assert matrice(t)
123     assert type(x) is int or type(x) is float
124
125     for i in range(len(t)):
126         for j in range(len(t[0])):
127             t[i][j] = x*t[i][j]
128
129     return t
130
131 # Transposée d'une matrice
132 def transposee(t):
133     """
134     s = transposee(t)
135     matrice transposée de la matrice t
136
137     >>> transposee([[1,2],[4,5]])
138     [[1, 4], [2, 5]]
139     >>> transposee([[1,2,3],[4,5,6]])
140     [[1, 4], [2, 5], [3, 6]]
141     """
142     assert matrice(t)
143
144     s = []
145     for j in range(len(t[0])):
146         s.append([])
147         for i in range(len(t)):
148             s[j].append(t[i][j])
149
150     return s
151
152 #-----
153 if __name__ == "__main__":
154     import doctest
155     doctest.testmod()

```

TD 4.26 : RECHERCHE D'UN MOTIF

```

1 # -*- coding: utf-8 -*-
2
3 def motif(t,m,debut,fin):
4     """
5     ok, i = motif(t,m,debut,fin)
6     ok == True : i est l'indice de la première occurrence
7     du motif m dans la liste t entre les indices debut et fin
8     ok == False : le motif m n'a pas été trouvé dans la liste t
9
10    >>> motif([1,2,3,2,3,4],[2,4],0,5)
11    (False, 5)
12    >>> motif([1,2,3,2,3,4],[2,3],0,5)
13    (True, 1)
14    >>> motif([1,2,3,2,3,4],[2,3],2,5)
15    (True, 3)
16    >>> motif([1,2,3,2,3,4],[2,3,4],0,5)
17    (True, 3)
18    """
19    assert type(t) is list
20    assert type(m) is list
21    assert 0 <= debut <= fin < len(t)
22
23    i,ok = debut,False
24    while i + len(m) - 1 <= fin and not ok:
25        if t[i:i+len(m)] == m and m != []:
26            ok = True
27        else: i = i + 1
28    return ok,i
29
30 #-----
31 if __name__ == "__main__":
32     import doctest
33     doctest.testmod()

```

TD 4.27 : RECHERCHE DE TOUTES LES OCCURENCES

```

1 # -*- coding: utf-8 -*-
2
3 def rechercheTout(t,x):
4     """
5     occurs = rechercheTout(t,x)
6     liste des indices de toutes les occurrences de x dans la liste t
7
8     >>> rechercheTout([1,2,1,5,1],1)
9     [0, 2, 4]
10    >>> rechercheTout([1,2,1,5,1],2)
11    [1]
12    >>> rechercheTout([1,2,1,5,1],3)
13    []
14    """
15    assert type(t) is list
16    occurs = []
17    for i in range(len(t)):
18        if t[i] == x: occurs.append(i)
19    return occurs
20

```

```

21 #-----
22 if __name__ == "__main__":
23     import doctest
24     doctest.testmod()

```

TD 4.28 : TRI BULLES

```

1 # -*- coding: utf-8 -*-
2
3 def triBulles(t,debut,fin):
4     """
5     t = triBulles(t,debut,fin)
6     trie la liste t entre les indices debut et fin
7     par la méthode du tri par bulles
8
9     >>> s = [9,8,7,6,5,4]
10    >>> triBulles(s,0,len(s)-1)
11    [4, 5, 6, 7, 8, 9]
12    >>> s = [9,8,7,6,5,4]
13    >>> triBulles(s,1,4)
14    [9, 5, 6, 7, 8, 4]
15    """
16    assert type(t) is list
17    assert 0 <= debut <= fin < len(t)
18
19    while debut < fin:
20        for j in range(fin,debut,-1):
21            if t[j] < t[j-1]:
22                t[j],t[j-1] = t[j-1],t[j]
23            debut = debut + 1
24
25    return t
26
27 #-----
28 if __name__ == "__main__":
29     import doctest
30     doctest.testmod()

```

TD 4.29 : MÉTHODE D'ÉLIMINATION DE GAUSS

```

1 # -*- coding: utf-8 -*-
2
3 from td425 import matriceCarree
4 from math import fabs
5
6 #-----
7 def solve(a,b):
8     """
9     x = solve(a,b)
10    x est la solution de l'équation matricielle a*x = b
11
12    >>> a, b = [[4]], [1]
13    >>> solve(a,b)
14    [0.25]
15    >>> a, b = [[1,1],[1,-1]], [1,0]
16    >>> solve(a,b)

```

```

17     [0.5, 0.5]
18     >>> a = [[2,-1,2],[1,10,-3],[-1,2,1]]
19     >>> b = [2,5,-3]
20     >>> solve(a,b)
21     [2.0, 0.0, -1.0]
22     """
23     assert matriceCarree(a)
24     assert type(b) is list
25     assert len(a) == len(b)
26
27     if triangularisation(a,b) == True:
28         x = backsubstitutions(a,b)
29     else: x = []
30
31     return x
32
33 #-----
34 def pivot(a,i0):
35     maxi = fabs(a[i0][i0])
36     r = i0
37     for i in range(i0+1,len(a)):
38         if fabs(a[i][i0]) > maxi:
39             maxi = fabs(a[i][i0])
40             r = i
41     return r
42
43 #-----
44 def subtractRows(a,b,k,i):
45     q = 1.*a[k][i]/a[i][i]
46     a[k][i] = 0
47     b[k] = b[k] - q*b[i]
48     for j in range(i+1,len(a)):
49         a[k][j] = a[k][j] - q*a[i][j]
50     return
51
52 #-----
53 def triangularisation(a,b):
54     ok = True; i = 0
55     while i < len(a) and ok == True:
56         p = pivot(a,i)
57         if i != p:
58             a[i],a[p] = a[p],a[i]
59             b[i],b[p] = b[p],b[i]
60         if a[i][i] == 0: ok = False
61         else:
62             for k in range(i+1,len(a)):
63                 subtractRows(a,b,k,i)
64         i = i + 1
65     return ok
66
67 #-----
68 def backsubstitutions(a,b):
69     n = len(a); x = []
70     for k in range(n): x.append(0)
71
72     x[n-1] = 1.*b[n-1]/a[n-1][n-1]
73     for i in range(n-2,-1,-1):
74         x[i] = b[i]
75         for j in range(i+1,n):

```



```

76         x[i] = x[i] - a[i][j]*x[j]
77         x[i] = 1.*x[i]/a[i][i]
78     return x
79
80 #-----
81 a = [[ 1,  0,  0, -1,  1,  0],\
82      [ 1,  1,  0,  0,  0,-1],\
83      [ 0,  1,-1,  0, -1,  0],\
84      [10,-10,  0,  0,-10,  0],\
85      [ 0,  0,  5,-20,-10,  0],\
86      [ 0, 10,  5,  0,  0,10]]
87 b = [0,0,0,0,0,12]
88 print(a,b)
89 print(solve(a,b))
90 print()
91
92 a, b = [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]], [32,23,33,31]
93 print(a,b)
94 print(solve(a,b))
95 print()
96
97 a, b = [[10,7,8.1,7.2],[7.08,5.04,6,5],[8,5.98,9.89,9],[6.99,4.99,9,9.98]], [32,23,33,31]
98 print(a,b)
99 print(solve(a,b))
100 print()
101
102 a, b = [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]], [32.01,22.99,33.01,30.99]
103 print(a,b)
104 print(solve(a,b))
105 print()
106
107
108 #-----
109 if __name__ == "__main__":
110     import doctest
111     doctest.testmod()

```

TD 4.30 : COMPARAISON D'ALGORITHMES DE RECHERCHE.

```

1 # -*- coding: utf-8 -*-
2
3 def mesureRecherche(f,t,x):
4     t0 = time()
5     f(tmp,x,0,len(t)-1)
6     dt = time() - t0
7     return dt
8
9 """
10 >>> s = liste(1000)
11 >>> x = s[len(s)-1]
12 >>> mesureRecherche(rechercheSequentielle,s,x)
13 0.00062489509582519531
14 >>> s = liste(100000)
15 >>> x = s[len(s)-1]
16 >>> mesureRecherche(rechercheSequentielle,s,x)
17 0.046545028686523438
18 """

```

TD 4.31 : COMPARAISON D'ALGORITHMES DE TRI

```
1 # -*- coding: utf-8 -*-
2
3 def mesureTri(f,t):
4     tmp = [x for x in t]
5     t0 = time()
6     f(tmp,0,len(t)-1)
7     dt = time() - t0
8     return dt
9
10 """
11 >>> s = liste(10000)
12 >>> mesureTri(triSelection,s)
13 23.040315866470337
14 >>> mesureTri(triInsertion,s)
15 22.086866855621338
16 >>> mesureTri(triRapide,s)
17 0.24324798583984375
18 """
```

Liste des exercices

1.1 Dessins sur la plage : exécution (1)	5
1.2 Dessins sur la plage : conception (1)	5
1.3 Propriétés d'un algorithme	6
1.4 Unités d'information	6
1.5 Première utilisation de PYTHON	7
1.6 Erreur de syntaxe en PYTHON	8
1.7 Dessins sur la plage : persévérance	8
1.8 Autonomie	8
1.9 Site WEB d'Informatique S1	9
1.10 Exemple de contrôle d'attention (1)	9
1.11 Exemple de contrôle de TD	9
1.12 Exemple de contrôle d'autoformation (1)	10
1.13 Exemple de contrôle des compétences	10
1.14 Nombre de contrôles	10
1.15 Exemple de contrôle d'autoformation (2)	11
1.16 Exemple de contrôle d'attention (2)	11
1.17 Nombres d'exercices de TD	12
1.18 Environnement de travail	12
1.19 QCM (1)	12
1.20 Puissance de calcul	12
1.21 Stockage de données	13
1.22 Dessins sur la plage : exécution (2)	13
1.23 Dessins sur la plage : conception (2)	13
1.24 Tracés de polygones réguliers	14
1.25 La multiplication « à la russe »	15
1.26 La multiplication arabe	16
1.27 La division chinoise	16
1.28 Le calcul Shadok	16
2.1 Unité de pression	19
2.2 Suite arithmétique (1)	19
2.3 Permutation circulaire (1)	19
2.4 Séquence d'affectations (1)	19
2.5 Opérateurs booléens dérivés (1)	20
2.6 Circuit logique (1)	20
2.7 Lois de De Morgan	20
2.8 Maximum de 2 nombres	21
2.9 Fonction « porte »	21
2.10 Ouverture d'un guichet	21

2.11	Catégorie sportive	22
2.12	Dessin d'étoiles (1)	22
2.13	Fonction factorielle	22
2.14	Fonction sinus	22
2.15	Algorithme d'Euclide	23
2.16	Division entière	23
2.17	Affichage inverse	23
2.18	Parcours inverse	23
2.19	Suite arithmétique (2)	23
2.20	Dessin d'étoiles (2)	24
2.21	Opérateurs booléens dérivés (2)	24
2.22	Damier	25
2.23	Trace de la fonction factorielle	25
2.24	Figure géométrique	26
2.25	Suite arithmétique (3)	26
2.26	QCM (2)	26
2.27	Unité de longueur	26
2.28	Permutation circulaire (2)	27
2.29	Séquence d'affectations (2)	27
2.30	Circuits logiques (2)	27
2.31	Alternative simple et test simple	28
2.32	Racines du trinôme	28
2.33	Séquences de tests	29
2.34	Racine carrée entière	30
2.35	Exécutions d'instructions itératives	30
2.36	Figures géométriques	31
2.37	Suites numériques	32
2.38	Calcul vectoriel	32
2.39	Prix d'une photocopie	33
2.40	Calcul des impôts	33
2.41	Développements limités	33
2.42	Tables de vérité	35
2.43	Dessins géométriques	36
2.44	Police d'assurance	36
2.45	Zéro d'une fonction	37
3.1	Codage des entiers positifs (1)	39
3.2	Codage d'un nombre fractionnaire	39
3.3	Décodage base $b \rightarrow$ décimal	40
3.4	Codage des entiers positifs (2)	41
3.5	Une spécification, des implémentations	41
3.6	Passage par valeur	42
3.7	Valeurs par défaut	43
3.8	Portée des variables	44
3.9	Tours de Hanoï à la main	44
3.10	Pgcd et ppcm de 2 entiers (1)	45
3.11	Somme arithmétique	46
3.12	Courbes fractales	48
3.13	QCM (3)	49

3.14	Passage des paramètres	49
3.15	Portée des variables (2)	50
3.16	Suite géométrique	51
3.17	Puissance entière	52
3.18	Coefficients du binôme	52
3.19	Fonction d'Ackerman	53
3.20	Addition binaire	54
3.21	Complément à 2	54
3.22	Codage-décodage des réels	56
3.23	Intégration numérique	61
3.24	Tracés de courbes paramétrées	63
4.1	Distance de 2 points de l'espace	67
4.2	Opérations sur les n-uplets	67
4.3	Pgcd et ppcm de 2 entiers (2)	68
4.4	Opérations sur les chaînes	69
4.5	Inverser une chaîne	71
4.6	Caractères, mots, lignes d'une chaîne	71
4.7	Opérations sur les listes (1)	72
4.8	Opérations sur les listes (2)	74
4.9	Sélection d'éléments	75
4.10	Opérations sur les piles	75
4.11	Opérations sur les files	77
4.12	Produit de matrices	78
4.13	Annuaire téléphonique	80
4.14	Recherche dichotomique	81
4.15	Liste ordonnée	83
4.16	Tri d'un annuaire téléphonique	84
4.17	Complexité du tri par sélection	85
4.18	Tri par insertion	86
4.19	Comparaison d'algorithmes (1)	87
4.20	QCM (4)	88
4.21	Génération de séquences	89
4.22	Application d'une fonction à tous les éléments d'une liste	89
4.23	Que fait cette procédure ?	90
4.24	Codes ASCII et chaînes de caractères	90
4.25	Opérations sur les matrices	91
4.26	Recherche d'un motif	93
4.27	Recherche de toutes les occurrences	94
4.28	Tri bulles	95
4.29	Méthode d'élimination de GAUSS	95
4.30	Comparaison d'algorithmes de recherche.	97
4.31	Comparaison d'algorithmes de tri	98