

Initiation à l'algorithmique


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 SEPTSEULT.

Ce document regroupe l'ensemble 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 ».



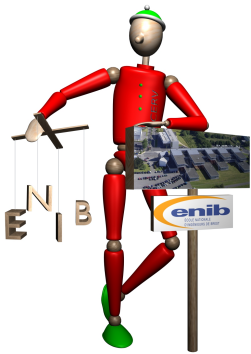
— Cours d'Informatique S1 —

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.

version du 21 octobre 2014



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

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

Sommaire

1	Introduction générale	5
2	Instructions de base	15
3	Procédures et fonctions	29
4	Structures linéaires	39
5	Annexes	47
5.1	Codes ASCII	47
5.2	Instructions PYTHON	48
5.3	Fonctions en PYTHON	49
5.4	Fonctions PYTHON prédéfinies	50
5.5	Instructions LOGO	51
5.6	Les séquences en PYTHON	51
5.7	Les fichiers en PYTHON	54
5.8	Utilitaire pydoc	55
5.9	Transformation d'une récursivité terminale	56
5.10	Méthode d'élimination de GAUSS	57
	Liste des exercices	63

Chapitre 1

Introduction générale

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

On cherche à faire dessiner une figure géométrique sur la plage à quelqu'un qui a les yeux bandés.

Quelle figure géométrique dessine-t-on en exécutant la suite d'instructions ci-dessous ?

1. avance de 10 pas,
2. tourne à gauche d'un angle de 120° ,
3. avance de 10 pas,
4. tourne à gauche d'un angle de 120° ,
5. avance de 10 pas.

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

Faire dessiner une spirale rectangulaire de 5 côtés, le plus petit côté faisant 2 pas de long et chaque côté fait un pas de plus que le précédent.

TD 1.3 : PROPRIÉTÉS D'UN ALGORITHME

Reprendre le TD 1.1 et illustrer la validité, la robustesse, la réutilisabilité, la complexité et l'efficacité de l'algorithme proposé pour dessiner sur la plage.

TD 1.4 : UNITÉS D'INFORMATION

Combien y a-t-il d'octets dans 1 ko (kilooctet), 1 Go (gigaoctet), 1 To (téraoctet), 1 Po (pétaoctet), 1 Eo (exaoctet), 1 Zo (zettaoctet) et 1 Yo (yottaoctet) ?

TD 1.5 : PREMIÈRE UTILISATION DE PYTHON

Se connecter sur un poste de travail d'une salle informatique.

1. Lancer PYTHON.
2. Utiliser PYTHON comme une simple calculatrice.
3. Quitter PYTHON.

Ne pas oublier de se déconnecter du poste de travail.

TD 1.6 : ERREUR DE SYNTAXE EN PYTHON

On considère la session PYTHON suivante :

```
>>> x = 3
>>> y = x
      File "<stdin>", line 1
        y = x
        ^
SyntaxError: invalid syntax
>>>
```


3. `a and (not b)`

4. `(not a) or b`

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

Répondre de mémoire aux questions suivantes (ie. sans rechercher les solutions dans les pages précédentes).

1. Quels sont les 4 types de contrôle proposés ?
2. Quels sont les documents que l'on peut trouver sur le site WEB du cours ?

TD 1.17 : NOMBRES D'EXERCICES DE TD

Combien d'exercices y avait-il à faire avant celui-ci ?

TD 1.18 : ENVIRONNEMENT DE TRAVAIL

Sur un poste de travail d'une salle informatique :

1. Quel est le type de clavier ?
2. Comment ouvre-t-on un terminal ?
3. Comment lance-t-on PYTHON ?
4. Où sont stockés les fichiers de travail ?

TD 1.19 : QCM (1)

(un seul item correct par question)

1. L'informatique est la science
 - (a) des dispositifs dont le fonctionnement dépend de la circulation d'électrons
 - (b) des signaux électriques porteurs d'information ou d'énergie
 - (c) du traitement automatique de l'information
 - (d) de la commande des appareils fonctionnant sans intervention humaine
2. Le logiciel est
 - (a) la mémoire de l'ordinateur
 - (b) le traitement automatique de l'information
 - (c) l'ensemble des données manipulées par les instructions
 - (d) un ensemble structuré d'instructions décrivant un traitement d'informations à faire réaliser par un matériel informatique
3. L'algorithmique est la science
 - (a) du traitement automatique de l'information
 - (b) des algorithmes
 - (c) des langages de programmation
 - (d) des instructions
4. Un algorithme est
 - (a) un ensemble de programmes remplissant une fonction déterminée, permettant l'accomplissement d'une tâche donnée
 - (b) une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents

- (c) le nombre d'instructions élémentaires à exécuter pour réaliser une tâche donnée
 - (d) un ensemble de dispositifs physiques utilisés pour traiter automatiquement des informations
5. La validité d'un algorithme est son aptitude
- (a) à utiliser de manière optimale les ressources du matériel qui l'exécute
 - (b) à se protéger de conditions anormales d'utilisation
 - (c) à calculer le nombre d'instructions élémentaires nécessaires pour réaliser la tâche pour laquelle il a été conçu
 - (d) à réaliser exactement la tâche pour laquelle il a été conçu
6. La complexité d'un algorithme est
- (a) le nombre de fois où l'algorithme est utilisé dans un programme
 - (b) le nombre de données manipulées par les instructions de l'algorithme
 - (c) le nombre d'octets occupés en mémoire par l'algorithme
 - (d) le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu
7. Un bit est
- (a) un chiffre binaire
 - (b) composé de 8 chiffres binaires
 - (c) un chiffre hexadécimal
 - (d) un mot d'un langage informatique
8. Un compilateur
- (a) exécute le code source
 - (b) exécute le bytecode
 - (c) traduit un code source en code objet
 - (d) exécute le code objet

TD 1.20 : PUISSANCE DE CALCUL

Donner l'ordre de grandeur en instructions par seconde des machines suivantes :

1. le premier micro-ordinateur de type PC,
2. une console de jeu actuelle,
3. un micro-ordinateur actuel,
4. Deeper-Blue : l'ordinateur qui a « battu » Kasparov aux échecs en 1997,
5. le plus puissant ordinateur actuel.

TD 1.21 : STOCKAGE DE DONNÉES

Donner l'ordre de grandeur en octets pour stocker en mémoire :

1. une page d'un livre,
2. une encyclopédie en 20 volumes,
3. une photo couleur,
4. une heure de vidéo,
5. une minute de son,

6. une heure de son.

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

1. Quelle figure géométrique dessine-t-on en exécutant la suite d'instructions ci-dessous ?

- (a) avance de 3 pas,
- (b) tourne à gauche d'un angle de 90° ,
- (c) avance de 4 pas,
- (d) rejoindre le point de départ.

2. Combien de pas a-t-on fait au total pour rejoindre le point de départ ?

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

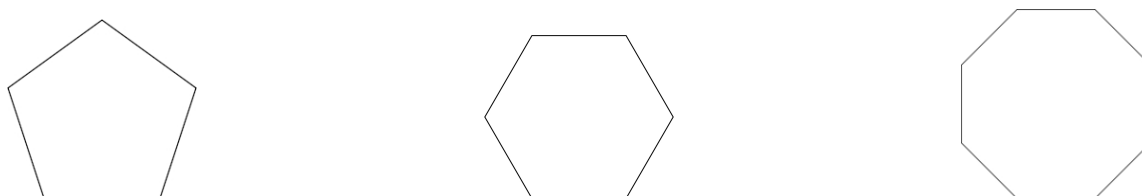
Reprendre le TD 1.2 et illustrer la validité, la robustesse, la réutilisabilité, la complexité et l'efficacité de l'algorithme proposé pour dessiner une spirale rectangulaire.

TD 1.24 : TRACÉS DE POLYGONES RÉGULIERS

On cherche à faire dessiner une figure polygonale (figure 1.1) sur la plage à quelqu'un qui a les yeux bandés. Pour cela, on ne dispose que de 2 commandes orales : avancer de n pas en avant (n est un nombre entier de pas) et tourner à gauche d'un angle θ (rotation sur place de θ).

- 1. Faire dessiner un pentagone régulier de 10 pas de côté.
- 2. Faire dessiner un hexagone régulier de 10 pas de côté.
- 3. Faire dessiner un octogone régulier de 10 pas de côté.
- 4. Faire dessiner un polygone régulier de n côtés de 10 pas chacun.

Fig. 1.1 : PENTAGONE, HEXAGONE, OCTOGONE



TD 1.25 : LA MULTIPLICATION « À LA RUSSE »

La technique de multiplication dite « à la russe » consiste à diviser par 2 le multiplicateur (et ensuite les quotients obtenus), jusqu'à un quotient nul, à noter les restes, et à multiplier parallèlement le multiplicande par 2. On additionne alors les multiples obtenus du multiplicande correspondant aux restes non nuls.

Exemple : $68 \times 123 (= 8364)$

<i>multiplicande</i> $M \times 2$	<i>multiplicateur</i> $m \div 2$	<i>reste</i> $m \bmod 2$	<i>somme partielle</i>
123	68	0	$(0 \times 123) + 0$
246	34	0	$(0 \times 246) + 0$
492	17	1	$(1 \times 492) + 0$
984	8	0	$(0 \times 984) + 492$
1968	4	0	$(0 \times 1968) + 492$
3936	2	0	$(0 \times 3936) + 492$
7872	1	1	$(1 \times 7872) + 492$
$68 \times 123 =$			8364

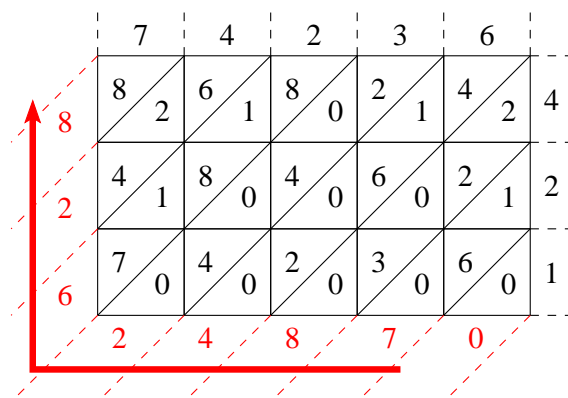
Effectuer les multiplications suivantes selon la technique « à la russe ».

1. $64 \times 96 (= 6144)$
2. $45 \times 239 (= 10755)$

TD 1.26 : LA MULTIPLICATION ARABE

On considère ici le texte d'Ibn al-Banna concernant la multiplication à l'aide de tableaux.

Fig. 1.2 : TABLEAU D'IBN AL-BANNA



« Tu construis un quadrilatère que tu subdivises verticalement et horizontalement en autant de bandes qu'il y a de positions dans les deux nombres multipliés. Tu divises diagonalement les carrés obtenus, à l'aide de diagonales allant du coin inférieur gauche au coin supérieur droit (figure 1.2).

Tu places le multiplicande au-dessus du quadrilatère, en faisant correspondre chacune de ses positions à une colonne¹. Puis, tu places le multiplicateur à gauche ou à droite du quadrilatère, de telle sorte qu'il descende avec lui en faisant correspondre également chacune de ses positions à une ligne². Puis, tu multiplies, l'une après l'autre, chacune des positions du multiplicande du carré par toutes les positions du multiplicateur, et tu poses le résultat partiel correspondant à chaque position dans le carré où se coupent respectivement leur colonne et leur ligne, en plaçant les unités au-dessus de la diagonale et les dizaines en dessous. Puis, tu commences à additionner, en partant du coin supérieur gauche : tu additionnes ce qui est entre les diagonales, sans effacer, en plaçant chaque nombre dans sa position, en transférant les dizaines de chaque somme partielle à la diagonale suivante et en les ajoutant à ce qui y figure.

1. L'écriture du nombre s'effectue de droite à gauche (exemple : 352 s'écrit donc 253).

2. L'écriture du nombre s'effectue de bas en haut (exemple : $\frac{3}{5}$ s'écrit donc $\frac{2}{5}$).

La somme que tu obtiendras sera le résultat. »

En utilisant la méthode du tableau d'Ibn al-Banna, calculer $63247 \times 124 (= 7842628)$.

TD 1.27 : LA DIVISION CHINOISE

Dans sa version actuelle, le boulier chinois se compose d'un nombre variable de tringles serties dans un cadre rectangulaire. Sur chacune de ces tringles, deux étages de boules séparées par une barre transversale peuvent coulisser librement (figure 1.3). La notation des nombres repose sur le principe de la numération de position : chacune des 2 boules du haut vaut 5 unités et chacune des 5 boules du bas vaut 1 unité. Seules comptent les boules situées dans la région transversale.

Il existe des règles spéciales de division pour chaque diviseur de 1 à 9. On considère ici les 7 règles de division par 7 (figure 1.4) :

1. « qi-yi xia jia san » : 7-1 ? ajouter 3 en dessous !
2. « qi-er xia jia liu » : 7-2 ? ajouter 6 en dessous !
3. « qi-san si sheng er » : 7-3 ? 4 reste 2 !
4. « qi-si wu sheng wu » : 7-4 ? 5 reste 5 !
5. « qi-wu qi sheng yi » : 7-5 ? 7 reste 1 !
6. « qi-liu ba sheng si » : 7-6 ? 8 reste 4 !
7. « feng-qi jin yi » : 7-7 ? 1 monté !

Fig. 1.3 : BOULIER CHINOIS

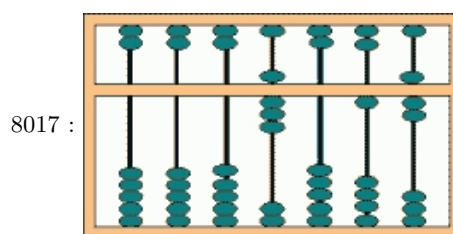


Fig. 1.4 : RÈGLES DE LA DIVISION PAR 7

Règle	Avant	Après
7-1	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 1 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 1 & 3 \end{array}$
7-2	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 2 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 0 & 1 \\ \hline 2 & 0 & 2 & 1 \end{array}$
7-3	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 3 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 4 & 2 \end{array}$
7-4	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 4 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ \hline 2 & 0 & 0 & 0 \end{array}$

Règle	Avant	Après
7-5	$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 2 & 0 & 0 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 2 & 0 & 2 & 1 \end{array}$
7-6	$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 2 & 0 & 1 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 2 & 0 & 3 & 4 \end{array}$
7-7	$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 2 & 0 & 2 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 1 & 0 & 0 \end{array}$

Ces règles ne sont pas des règles logiques, mais de simples procédés mnémotechniques indiquant ce qu'il convient de faire selon la situation. Leur énoncé débute par le rappel du diviseur, ici 7, et se poursuit par l'énoncé du dividende, par exemple 3 : 7-3. Le reste de la règle indique quelles manipulations effectuées, ajouts ou retraits de boules. Il faut également savoir que le dividende étant posé sur le boulier, on doit appliquer les règles aux chiffres successifs du dividende, en commençant par celui dont l'ordre est le plus élevé. « ajouter en dessous » veut dire « mettre

des boules au rang immédiatement inférieur (à droite) au rang considéré » et « monter » veut dire « mettre des boules au rang immédiatement supérieur (à gauche) au rang considéré ».

Pour effectuer la division d'un nombre par 7, on pose le dividende à droite sur le boulier et le diviseur (7) à gauche. On opère sur les chiffres successifs du dividende en commençant par celui d'ordre le plus élevé (le plus à gauche). Les règles précédemment énoncées sont appliquées systématiquement.

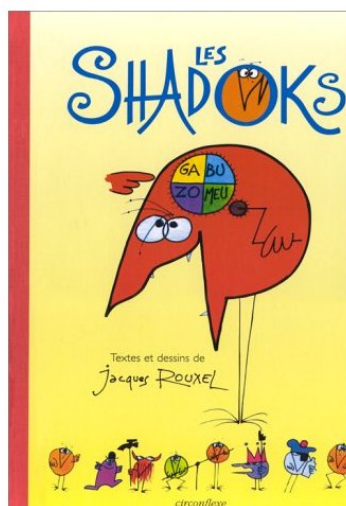
Utiliser un boulier chinois pour diviser 1234 par 7 ($1234 = 176 \times 7 + 2$).

$$\begin{array}{r} 1 \quad 0 \quad 0 \quad 0 \quad 0 \\ 2 \overline{) \quad} \end{array} \rightarrow \begin{array}{r} 1 \quad 0 \quad 1 \quad 1 \quad 0 \\ 2 \overline{) \quad} \end{array}$$

TD 1.28 : LE CALCUL SHADOK

Les cerveaux des Shadoks avaient une capacité tout à fait limitée. Ils ne comportaient en tout que 4 cases. Comme ils n'avaient que 4 cases, évidemment les Shadoks ne connaissaient pas plus de 4 mots : GA, BU, ZO ET MEU (figure 1.5).

Fig. 1.5 : LES SHADOKS : GA BU ZO MEU



Etant donné qu'avec 4 mots, ils ne pouvaient pas compter plus loin que 4, le Professeur Shadoko avait réformé tout ça :

- Quand il n'y a pas de Shadok, on dit GA et on écrit GA.
- Quand il y a un Shadok de plus, on dit BU et on écrit BU.
- Quand il y a encore un Shadok, on dit ZO et on écrit ZO.
- Et quand il y en a encore un autre, on dit MEU et on écrit MEU.

Tout le monde applaudissait très fort et trouvait ça génial sauf le Devin Plombier qui disait qu'on n'avait pas idée d'inculquer à des enfants des bêtises pareilles et que Shadoko, il fallait le condamner. Il fut très applaudi aussi. Les mathématiques, cela les intéressait, bien sûr, mais brûler le professeur, c'était intéressant aussi, faut dire. Il fut décidé à l'unanimité qu'on le laisserait parler et qu'on le brûlerait après, à la récréation.

- Répétez avec moi : MEU ZO BU GA...GA BU ZO MEU.
- Et après ! ricanait le Plombier.

- Si je mets un Shadok en plus, évidemment, je n'ai plus assez de mots pour les compter, alors c'est très simple : on les jette dans une poubelle, et je dis que j'ai BU poubelle. Et pour ne pas confondre avec le BU du début, je dis qu'il n'y a pas de Shadok à côté de la poubelle et j'écris BU GA. BU Shadok à côté de la poubelle : BU BU. Un autre : BU ZO. Encore un autre : BU MEU. On continue. ZO poubelles et pas de Shadok à côté : ZO GA...MEU poubelles et MEU Shadoks à côté : MEU MEU. Arrivé là, si je mets un Shadok en plus, il me faut une autre poubelle. Mais comme je n'ai plus de mots pour compter les poubelles, je m'en débarrasse en les jetant dans une grande poubelle. J'écris BU grande poubelle avec pas de petite poubelle et pas de Shadok à côté : BU GA GA, et on continue...BU GA BU, BU GA ZO...MEU MEU ZO, MEU MEU MEU. Quand on arrive là et qu'on a trop de grandes poubelles pour pouvoir les compter, eh bien, on les met dans une super-poubelle, on écrit BU GA GA GA, et on continue...(figure 1.6).

Fig. 1.6 : LES 18 PREMIERS NOMBRES SHADOK

0 : GA	6 : BU ZO	12 : MEU GA
1 : BU	7 : BU MEU	13 : MEU BU
2 : ZO	8 : ZO GA	14 : MEU ZO
3 : MEU	9 : ZO BU	15 : MEU MEU
4 : BU GA	10 : ZO ZO	16 : BU GA GA
5 : BU BU	11 : ZO MEU	17 : BU GA BU

- Quels sont les entiers décimaux représentés en « base Shadok » par les expressions suivantes ?
 - GA GA
 - BU BU BU
 - ZO ZO ZO ZO
 - MEU MEU MEU MEU MEU
- Effectuer les calculs Shadok suivants.
 - ZO ZO MEU + BU GA MEU
 - MEU GA MEU – BU MEU GA
 - ZO MEU MEU × BU GA MEU
 - ZO ZO ZO MEU ÷ BU GA ZO

Chapitre 2

Instructions de base

TD 2.1 : UNITÉ DE PRESSION

Le torr (torr) ou millimètre de mercure (mmHg) est une unité de mesure de la pression qui tire son nom du physicien et mathématicien italien Evangelista Torricelli (1608-1647). Il est défini comme la pression exercée à 0°C par une colonne de 1 millimètre de mercure (mmHg). Il a plus tard été indexée sur la pression atmosphérique : 1 atmosphère normale correspond à 760 mmHg et à 101 325 Pa.

Ecrire une instruction qui permette de passer directement des torrs au pascals (Pa).

TD 2.2 : SUITE ARITHMÉTIQUE (1)

Ecrire une instruction qui calcule la somme $s = \sum_0^n u_k$ des n premiers termes d'une suite arithmétique $u_k = a + r \cdot k$.

TD 2.3 : PERMUTATION CIRCULAIRE (1)

Effectuer une permutation circulaire droite entre les valeurs de 4 entiers x , y , z et t .

TD 2.4 : SÉQUENCE D'AFFECTIONS (1)

Quelles sont les valeurs des variables a , b , q et r après la séquence d'affectations suivante ?

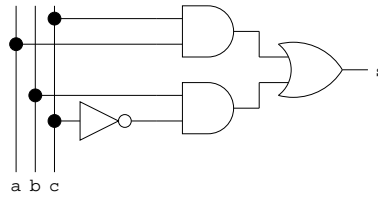
```
a = 19
b = 6
q = 0
r = a
r = r - b
q = q + 1
r = r - b
q = q + 1
r = r - b
q = q + 1
```

TD 2.5 : OPÉRATEURS BOOLÉENS DÉRIVÉS (1)

En utilisant les opérateurs booléens de base (**not**, **and** et **or**), écrire un algorithme qui affecte successivement à une variable **s** le résultat des opérations booléennes suivantes : ou exclusif (xor , $a \oplus b$), non ou (nor , $\overline{a + b}$), non et ($nand$, $\overline{a \cdot b}$), implication ($a \Rightarrow b$) et équivalence ($a \Leftrightarrow b$).

TD 2.6 : CIRCUIT LOGIQUE (1)

Donner les séquences d'affectations permettant de calculer la sortie s du circuit logique suivant en fonction de ses entrées a , b et c .

**TD 2.7 : LOIS DE DE MORGAN**

Démontrer à l'aide des tables de vérité les lois de De Morgan $\forall a, b \in \{0; 1\}$:

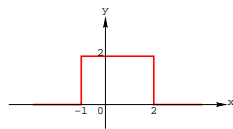
1. $\overline{(a + b)} = \bar{a} \cdot \bar{b}$
2. $\overline{(a \cdot b)} = \bar{a} + \bar{b}$

TD 2.8 : MAXIMUM DE 2 NOMBRES

Ecrire un algorithme qui détermine le maximum m de 2 nombres x et y .

TD 2.9 : FONCTION « PORTE »

Proposer une autre alternative simple pour calculer la fonction « porte » de l'exemple ci-dessous.

**TD 2.10 : OUVERTURE D'UN GUICHET**

A l'aide d'alternatives simples imbriquées, écrire un algorithme qui détermine si un guichet est 'ouvert' ou 'fermé' selon les jours de la semaine ('lundi', 'mardi', ... , 'dimanche') et l'heure de la journée (entre 0h et 24h). Le guichet est ouvert tous les jours de 8h à 13h et de 14h à 17h sauf le samedi après-midi et toute la journée du dimanche.

TD 2.11 : CATÉGORIE SPORTIVE

Ecrire un algorithme qui détermine la catégorie sportive d'un enfant selon son âge :

- Poussin de 6 à 7 ans,
- Pupille de 8 à 9 ans,
- Minime de 10 à 11 ans,
- Cadet de 12 ans à 14 ans.

TD 2.12 : DESSIN D'ÉTOILES (1)

Ecrire un algorithme itératif qui affiche les n lignes suivantes (l'exemple est donné ici pour $n = 6$) :

```

*****
*****
****
***
**
*
```

Rappel PYTHON :

```

>>> 5*'r'
'rrrrr'
>>> 2*'to'
'toto'
```

TD 2.13 : FONCTION FACTORIELLE

Ecrire un algorithme qui calcule $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$.

TD 2.14 : FONCTION SINUS

Ecrire un algorithme qui calcule de manière itérative la fonction sinus en fonction de son développement en série entière.

$$\sin(x) \approx \sum_{k=0}^n u_k = \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{6} + \frac{x^5}{120} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Les calculs seront arrêtés lorsque la valeur absolue du terme u_k sera inférieure à un certain seuil s ($0 < s < 1$). On n'utilisera ni la fonction *puissance* (x^n) ni la fonction *factorielle* ($n!$) pour effectuer le calcul de $\sin(x)$.

TD 2.15 : ALGORITHME D'EUCLIDE

Dans la tradition grecque, en comprenant un nombre entier comme une longueur, un couple d'entiers comme un rectangle, leur pgcd est la taille du plus grand carré permettant de carreler ce rectangle. L'algorithme décompose ce rectangle en carrés, de plus en plus petits, par divisions euclidiennes successives, de la longueur par la largeur, puis de la largeur par le reste, jusqu'à un reste nul.

Faire la construction géométrique « à la grecque antique » qui permet de déterminer le pgcd d de $a = 21$ et $b = 15$ ($d = 3$).

TD 2.16 : DIVISION ENTIÈRE

Ecrire un algorithme itératif qui calcule le quotient q et le reste r de la division entière $a \div b$ ($a = bq + r$).

On n'utilisera pas les opérateurs prédéfinis `/` et `%` mais on pourra s'inspirer du TD 2.4 page 15.

TD 2.17 : AFFICHAGE INVERSE

Ecrire un algorithme qui affiche les caractères d'une chaîne s , un par ligne en partant de la fin de la chaîne.

TD 2.18 : PARCOURS INVERSE

Ecrire un algorithme qui parcourt en sens inverse une séquence s quelconque (du dernier élément au premier élément).

TD 2.19 : SUITE ARITHMÉTIQUE (2)

1. Ecrire un algorithme qui calcule de manière itérative la somme $s = \sum_0^n u_k$ des n premiers termes d'une suite arithmétique $u_k = a + r \cdot k$. On utilisera une boucle `for`.
2. Comparer l'efficacité de cette approche itérative avec le calcul du TD 2.2 page 15.

TD 2.20 : DESSIN D'ÉTOILES (2)

Reprendre le TD 2.12 page 16 en supposant qu'on ne peut afficher qu'une étoile à la fois (on s'interdit ici la possibilité d'écrire `5*'*'` à la place de `'*****'` par exemple).

TD 2.21 : OPÉRATEURS BOOLÉENS DÉRIVÉS (2)

A l'aide d'itérations imbriquées, afficher les tables de vérité des opérateurs logiques dérivés (voir TD 2.5) : ou exclusif ($xor, a \oplus b$), non ou ($nor, \overline{a + b}$), non et ($nand, \overline{a \cdot b}$), implication ($a \Rightarrow b$) et équivalence ($a \Leftrightarrow b$).

TD 2.22 : DAMIER

En utilisant les instructions à la LOGO de l'annexe 5.5 page 51, dessiner un damier rectangulaire de $n \times m$ cases.

TD 2.23 : TRACE DE LA FONCTION FACTORIELLE

Tracer la fonction factorielle du TD 2.13 page 16.

TD 2.24 : FIGURE GÉOMÉTRIQUE

Que dessinent les instructions suivantes ?

```
x0 = 0
y0 = 0
r = 10
n = 5
m = 10
for i in range(n) :
    up()
    y = y0 - 2*r*i
    x = x0 + r*(i%2)
    goto(x,y)
    for j in range(m) :
        down()
        circle(r)
        up()
        x = x + 2*r
        goto(x,y)
```

TD 2.25 : SUITE ARITHMÉTIQUE (3)

Reprendre le TD 2.19 page 17 en explicitant l'invariant, la condition d'arrêt, la progression et l'initialisation de la boucle retenue.

TD 2.26 : QCM (2)

(un seul item correct par question)

1. En PYTHON, l'instruction « ne rien faire » se dit
 - (a) `break`
 - (b) `return`
 - (c) `pass`
 - (d) `continue`
2. Une variable informatique est un objet
 - (a) équivalent à une variable mathématique
 - (b) qui associe un nom à une valeur
 - (c) qui varie nécessairement
 - (d) qui modifie la mémoire
3. L'affectation consiste à
 - (a) comparer la valeur d'une variable à une autre valeur
 - (b) associer une valeur à une variable
 - (c) incrémenter une variable
 - (d) déplacer une variable en mémoire

4. Après la séquence

a = 13
b = 4
b = a
a = b

les variables a et b sont telles que

- (a) $a = 13$ et $b = 13$
 - (b) $a = 4$ et $b = 4$
 - (c) $a = 4$ et $b = 13$
 - (d) $a = 13$ et $b = 4$
5. Le résultat d'une comparaison est une valeur
- (a) réelle
 - (b) qui dépend du type des arguments
 - (c) booléenne
 - (d) entière
6. Un opérateur booléen s'applique à des valeurs
- (a) booléennes
 - (b) entières
 - (c) réelles
 - (d) alphanumériques
7. La fonction principale d'une instruction de test est
- (a) de passer d'instruction en instruction
 - (b) de répéter une instruction sous condition
 - (c) d'exécuter une instruction sous condition
 - (d) d'interrompre l'exécution d'une instruction

8. Après la séquence

```
x = -3
if x < -4 : y = 0
elif x < -3 : y = 4 - x
elif x < -1 : y = x*x + 6*x + 8
elif x < 3 : y = 2 - x
else : y = -2
```

la variable y est telle que

- (a) $y = -1$
 - (b) $y = 0$
 - (c) $y = 7$
 - (d) $y = -2$
9. L'itération conditionnelle est une instruction de contrôle du flux d'instructions
- (a) qui permet d'exécuter une instruction sous condition préalable.
 - (b) qui est vérifiée tout au long de son exécution.
 - (c) qui permet sous condition préalable de répéter zéro ou plusieurs fois la même instruction.
 - (d) qui permet de choisir entre plusieurs instructions.
10. On ne sort jamais d'une boucle si la condition d'arrêt
- (a) ne varie pas en cours d'exécution.
 - (b) ne contient pas d'opérateurs booléens.
 - (c) est toujours fausse.
 - (d) n'est jamais fausse.

11. Que vaut `f` à la fin des instructions suivantes si $n = 5$?

```
f = 0
i = 1
while i < n+1:
    f = f + i
    i = i + 1
```

- (a) 6
 - (b) 10
 - (c) 15
 - (d) 21
12. Une séquence est une suite ordonnée
- (a) d'éléments que l'on peut référencer par leur rang.
 - (b) d'instructions formant un ensemble logique.
 - (c) d'instructions conditionnelles.
 - (d) de nombres
13. Dans la chaîne `s = 'gérard'`, `s[2]` vaut
- (a) 'é'
 - (b) 'r'
 - (c) 'gé'
 - (d) 'gér'
14. Que vaut `f` à la fin des instructions suivantes si $n = 5$?

```
f = 1
for i in range(2,n+1) :
    f = f * i
```

- (a) 120
 - (b) 720
 - (c) 6
 - (d) 24
15. Que vaut `f` à la fin des instructions suivantes si $n = 5$?

```
f, f1, f2 = 2,1,1
for i in range(3,n+1) :
    f2 = f1
    f1 = f
    f = f1 + f2
```

- (a) 3
- (b) 5
- (c) 8
- (d) 13

TD 2.27 : UNITÉ DE LONGUEUR

L'année-lumière (al) est une unité de distance utilisée en astronomie. Une année-lumière est la distance parcourue par un photon (ou plus simplement la lumière) dans le vide, en dehors de tout champ gravitationnel ou magnétique, en une année julienne (365,25 jours).

Ecrire une instruction qui permette de passer directement des années-lumière aux m/s sachant que la vitesse de la lumière dans le vide est de 299 792 458 m/s.

TD 2.28 : PERMUTATION CIRCULAIRE (2)

Effectuer une permutation circulaire gauche entre les valeurs de 3 entiers x , y et z .

TD 2.29 : SÉQUENCE D'AFFECTATIONS (2)

Quelles sont les valeurs des variables n et s après la séquence d'affectations suivante ?

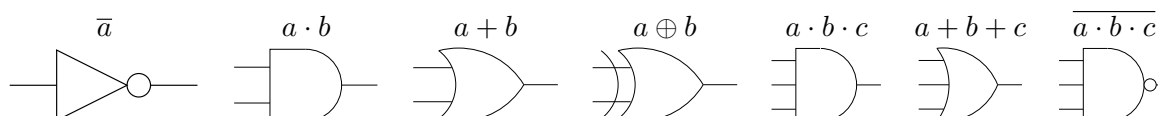
```

n = 1
s = n
n = n + 1
s = s + n
n = n + 1
s = s + n
n = n + 1
s = s + n
n = n + 1
s = s + n

```

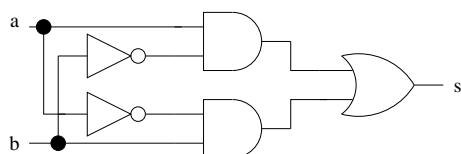
TD 2.30 : CIRCUITS LOGIQUES (2)

On considère les conventions graphiques traditionnelles pour les opérateurs logiques :

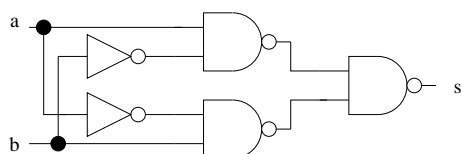


Donner les séquences d'affectations permettant de calculer la (ou les) sortie(s) des circuits logiques suivants en fonction de leurs entrées.

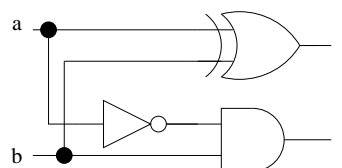
1. a et b sont les entrées, s la sortie.



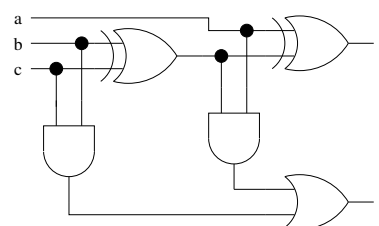
2. a et b sont les entrées, s la sortie.



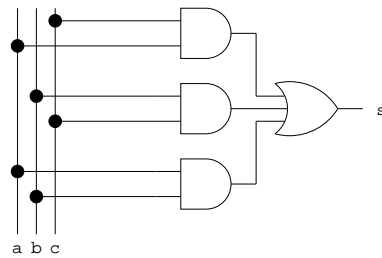
3. a et b sont les entrées, s et t les sorties.



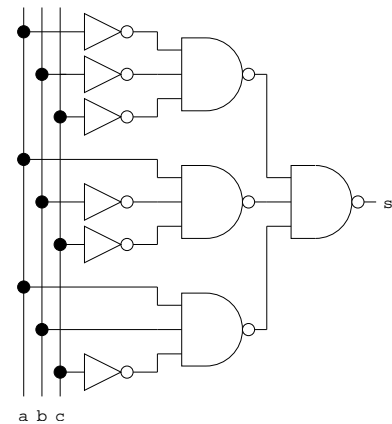
4. a , b et c sont les entrées, s et t les sorties.



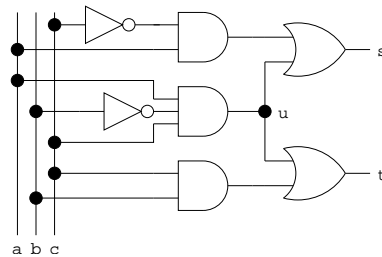
5. a , b et c sont les entrées et s la sortie.



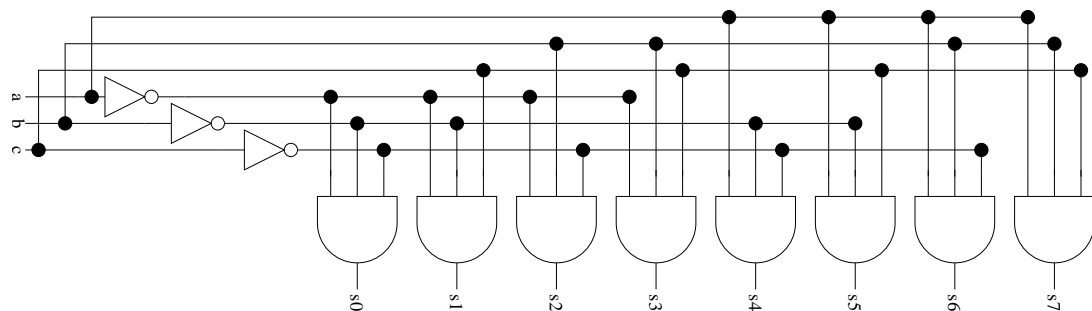
7. a , b et c sont les entrées, s la sortie.



6. a , b et c sont les entrées, s et t les sorties.



8. a , b et c sont les entrées, $s_0, s_1 \dots s_7$ les sorties.



TD 2.31 : ALTERNATIVE SIMPLE ET TEST SIMPLE

Montrer à l'aide d'un contre-exemple que l'alternative simple :

```
if condition : blocIf
else : blocElse
```

n'est pas équivalente à la séquence de tests simples suivante :

```
if condition : blocIf
if not condition : blocElse
```

TD 2.32 : RACINES DU TRINOME

Ecrire un algorithme qui calcule les racines x_1 et x_2 du trinome $ax^2 + bx + c$.

TD 2.33 : SÉQUENCES DE TESTS

1. Quelle est la valeur de la variable x après la suite d'instructions suivante ?

```
x = -3
if x < 0 : x = -x
```

2. Quelle est la valeur de la variable y après la suite d'instructions suivante ?

```
x0 = 3
x = 5
if x < x0 : y = -1
else : y = 1
```

3. Quelle est la valeur de la variable y après la suite d'instructions suivante ?

```
p = 1
d = 0
r = 0
h = 1
z = 0
f = p and (d or r)
g = not r
m = not p and not z
g = g and (d or h or m)
if f or g : y = 1
else : y = 0
```

4. Quelle est la valeur de la variable ok après la suite d'instructions suivante ?

```
x = 2
y = 3
d = 5
h = 4
if x > 0 and x < d :
    if y > 0 and y < h : ok = 1
    else : ok = 0
else : ok = 0
```

5. Quelle est la valeur de la variable y après la suite d'instructions suivante ?

```
x = 3
y = -2
if x < y : y = y - x
elif x == y : y = 0
else : y = x - y
```

TD 2.34 : RACINE CARRÉE ENTIÈRE

Ecrire un algorithme qui calcule la racine carrée entière r d'un nombre entier positif n telle que $r^2 \leq n < (r+1)^2$.

TD 2.35 : EXÉCUTIONS D'INSTRUCTIONS ITÉRATIVES

1. Que fait cette suite d'instructions ?

```
x = 0
while x != 33 :
    x = input('entrer un nombre : ')
```

2. Que fait cette suite d'instructions ?

```
x = 0
while x <= 0 or x > 5 :
    x = input('entrer un nombre : ')
```

3. Que fait cette suite d'instructions ?

```
s = 0
for i in range(5) :
    x = input('entrer un nombre : ')
    s = s + x
```

4. Qu'affichent les itérations suivantes ?

```
for i in range(0,10) :
    for j in range(0,i) :
        print('*',end=' ')
    print()
```

5. Qu'affichent les itérations suivantes ?

```
for i in range(0,10) :
    j = 10 - i
    while j > 0 :
        print('*',end=' ')
        j = j - 1
    print()
```

6. Qu'affichent les itérations suivantes ?

```
for i in range(1,10) :
    for j in range(0,11) :
        print(i,'x',j,' = ',i*j)
    print()
```

7. Qu'affichent les itérations suivantes ?

```
for n in range(10) :
    for p in range(n+1) :
        num = 1
        den = 1
        for i in range(1,p+1) :
            num = num*(n-i+1)
            den = den*i
        c = num/den
        print(c,end=' ')
    print()
```

8. Qu'affichent les itérations suivantes ?

```
for n in range(0,15) :
    f = 1
    f1 = 1
    f2 = 1
    for i in range(2,n+1) :
        f = f1 + f2
        f2 = f1
        f1 = f
    print(f,end=' ')
```

9. Quelle est la valeur de la variable s à la fin des instructions suivantes ?

```
b = 2
k = 8
n = 23
s = 0
i = k - 1
q = n
while q != 0 and i >= 0 :
    s = s + (q%b)*b**(k-1-i)
    print(q%b,end=' ')
    q = q/b
    i = i - 1
```

TD 2.36 : FIGURES GÉOMÉTRIQUES

1. Ecrire un algorithme qui calcule le périmètre p et la surface s d'un rectangle de longueur L et de largeur l .
2. Ecrire un algorithme qui calcule le périmètre p et la surface s d'un cercle de rayon r .
3. Ecrire un algorithme qui calcule la surface latérale s et le volume v d'un cylindre de rayon r et de hauteur h .
4. Ecrire un algorithme qui calcule la surface s et le volume v d'une sphère de rayon r .

TD 2.37 : SUITES NUMÉRIQUES

1. Ecrire un algorithme qui calcule la somme $s = \sum_0^n u_k$ des n premiers termes d'une suite arithmétique $u_k = a + bk$.
2. Ecrire un algorithme qui calcule la somme $s = \sum_0^n u_k$ des n premiers termes d'une suite géométrique $u_k = ab^k$.

TD 2.38 : CALCUL VECTORIEL

1. Ecrire un algorithme qui calcule le module r et les cosinus directeurs a , b et c d'un vecteur de composantes (x, y, z) .
2. Ecrire un algorithme qui calcule le produit scalaire p de 2 vecteurs de composantes respectives (x_1, y_1, z_1) et (x_2, y_2, z_2) .
3. Ecrire un algorithme qui calcule les composantes (x_3, y_3, z_3) du produit vectoriel de 2 vecteurs de composantes respectives (x_1, y_1, z_1) et (x_2, y_2, z_2) .
4. Ecrire un algorithme qui calcule le produit mixte v de 3 vecteurs de composantes respectives (x_1, y_1, z_1) , (x_2, y_2, z_2) et (x_3, y_3, z_3) .

TD 2.39 : PRIX D'UNE PHOTOCOPIE

Ecrire un algorithme qui affiche le prix de n photocopies sachant que le reprographe facture 0,10 E les dix premières photocopies, 0,09 E les vingt suivantes et 0,08 E au-delà.

TD 2.40 : CALCUL DES IMPÔTS

Ecrire un algorithme qui affiche si un contribuable d'un pays imaginaire est imposable ou non sachant que :

- les hommes de plus de 18 ans paient l'impôt,
- les femmes paient l'impôt si elles ont entre 18 et 35 ans,
- les autres ne paient pas d'impôt.

TD 2.41 : DÉVELOPPEMENTS LIMITÉS

Calculer chaque fonction ci-dessous en fonction de son développement en série entière ($\sum u_k$). Les calculs seront arrêtés lorsque la valeur absolue du terme u_k sera inférieure à un certain seuil s ($0 < s < 1$).

On n'utilisera ni la fonction *puissance* (x^n) ni la fonction *factorielle* ($n!$).

1. $\sinh(x) \approx \sum_{k=0}^n \frac{x^{2k+1}}{(2k+1)!} = x + \frac{x^3}{6} + \frac{x^5}{120} + \dots + \frac{x^{2n+1}}{(2n+1)!}$
2. $\cosh(x) \approx \sum_{k=0}^n \frac{x^{2k}}{(2k)!} = 1 + \frac{x^2}{2} + \frac{x^4}{24} + \dots + \frac{x^{2n}}{(2n)!}$
3. $\cos(x) \approx \sum_{k=0}^n (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2} + \frac{x^4}{24} + \dots + (-1)^n \frac{x^{2n}}{(2n)!}$
4. $\log(1+x) \approx \sum_{k=0}^n (-1)^k \frac{x^{k+1}}{k+1} = x - \frac{x^2}{2} + \frac{x^3}{3} + \dots + (-1)^n \frac{x^{n+1}}{n+1}$, pour $-1 < x < 1$
5. $\arctan(x) \approx \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)}$, pour $-1 < x < 1$

TD 2.42 : TABLES DE VÉRITÉ

A l'aide d'itérations imbriquées, afficher les tables de vérité des circuits logiques du TD 2.30 page 21.

TD 2.43 : DESSINS GÉOMÉTRIQUES

1. Que dessine la suite d'instructions suivante ?

```
forward(20)
right(144)
forward(20)
right(144)
forward(20)
right(144)
forward(20)
right(144)
forward(20)
right(144)
```

2. Que dessine la suite d'instructions suivante ?

```
forward(10)
left(45)
forward(10)
left(135)
forward(10)
left(45)
forward(10)
left(135)
```

TD 2.44 : POLICE D'ASSURANCE

Une compagnie d'assurance automobile propose 4 familles de tarifs du moins cher au plus onéreux : A, B, C et D. Le tarif dépend de la situation du conducteur.

- Un conducteur de moins de 25 ans et titulaire du permis depuis moins de deux ans, se voit attribuer le tarif D s'il n'a jamais été responsable d'accident. Sinon, la compagnie refuse de l'assurer.
- Un conducteur de moins de 25 ans et titulaire du permis depuis plus de deux ans, ou de plus de 25 ans mais titulaire du permis depuis moins de deux ans a le droit au tarif C s'il n'a jamais provoqué d'accident, au tarif D pour un accident, sinon il est refusé.
- Un conducteur de plus de 25 ans titulaire du permis depuis plus de deux ans bénéficie du tarif B s'il n'est à l'origine d'aucun accident et du tarif C pour un accident, du tarif D pour deux accidents, et refusé sinon.

Par ailleurs, pour encourager la fidélité de ses clients, la compagnie propose un contrat au tarif immédiatement inférieur s'il est assuré depuis plus d'un an.

Ecrire un algorithme qui propose un tarif d'assurance selon les caractéristiques d'un client potentiel.

TD 2.45 : ZÉRO D'UNE FONCTION

On recherche le zéro d'une fonction f continue sur un intervalle $[a, b]$ telle que $f(a).f(b) < 0$; il existe donc une racine de f dans $]a, b[$ que nous supposons unique.

1. Ecrire un algorithme qui détermine le zéro de $\cos(x)$ dans $[1, 2]$ selon la méthode par dichotomie.

Indications : on pose $x_1 = a$, $x_2 = b$ et $x = (x_1 + x_2)/2$. Si $f(x_1).f(x) < 0$, la racine est dans $]x_1, x[$ et on pose $x_2 = x$; sinon la racine est dans $]x, x_2[$ et on pose $x_1 = x$. Puis on réitère le procédé, la longueur de l'intervalle ayant été divisée par deux. Lorsque x_1 et x_2 seront suffisamment proches, on décidera que la racine est x .

2. Ecrire un algorithme qui détermine le zéro de $\cos(x)$ dans $[1, 2]$ selon la méthode des tangentes.

Indications : soit x_n une approximation de la racine c recherchée : $f(c) = f(x_n) + (c - x_n)f'(x_n)$; comme $f(c) = 0$, on a : $c = x_n - f(x_n)/f'(x_n)$. Posons $x_{n+1} = x_n - f(x_n)/f'(x_n)$: on peut considérer que x_{n+1} est une meilleure approximation de c que x_n . On recommence le procédé avec x_{n+1} et ainsi de suite jusqu'à ce que $|x_{n+1} - x_n|$ soit inférieur à un certain seuil s .

3. Ecrire un algorithme qui détermine le zéro de $\cos(x)$ dans $[1, 2]$ selon la méthode des sécantes.

Indications : reprendre la méthode des tangentes en effectuant l'approximation suivante :

$$f'(x_n) = (f(x_n) - f(x_{n-1})) / (x_n - x_{n-1}).$$

4. Ecrire un algorithme qui détermine le zéro de $\cos(x)$ dans $[1, 2]$ selon la méthode des cordes.

Indications : reprendre la méthode par dichotomie en prenant pour x le point d'intersection de la corde AB et de l'axe des abscisses : $x = (x_2 f(x_1) - x_1 f(x_2)) / (f(x_1) - f(x_2))$, c'est-à-dire le point obtenu par la méthode des sécantes.

Chapitre 3

Procédures et fonctions

TD 3.1 : CODAGE DES ENTIERS POSITIFS (1)

Définir un algorithme qui code sur k chiffres en base b un entier positif n du système décimal.

Exemples : $(38)_{10} \rightarrow (123)_5$
 $(83)_{10} \rightarrow (123)_8$
 $(291)_{10} \rightarrow (123)_{16}$

TD 3.2 : CODAGE D'UN NOMBRE FRACTIONNAIRE

Un nombre fractionnaire (*nombre avec des chiffres après la virgule* : $(r_n r_{n-1} \dots r_1 r_0 . r_{-1} r_{-2} \dots)_b$) est défini sur un sous-ensemble borné, incomplet et fini des rationnels. Un tel nombre a pour valeur :

$$r_n b^n + r_{n-1} b^{n-1} + \dots + r_1 b^1 + r_0 b^0 + r_{-1} b^{-1} + r_{-2} b^{-2} + \dots$$

En pratique, le *nombre de chiffres après la virgule* est limité par la taille physique en machine.

$$(r_n r_{n-1} \dots r_1 r_0 . r_{-1} r_{-2} \dots r_{-k})_b = \sum_{i=-k}^{i=n} r_i b^i$$

Un nombre x pourra être représenté en base b par un triplet $[s, m, p]$ tel que $x = (-1)^s \cdot m \cdot b^p$ où s représente le signe de x , m sa mantisse et p son exposant (p comme puissance) où :

- signe s : $s = 1$ si $x < 0$ et $s = 0$ si $x \geq 0$
- mantisse m : $m \in [1, b[$ si $x \neq 0$ et $m = 0$ si $x = 0$
- exposant p : $p \in [\min, \max]$

Ainsi, le codage de $x = -9.75$ en base $b = 2$ s'effectuera en 4 étapes :

1. coder le signe de x : $x = -9.75 < 0 \Rightarrow s = 1$
2. coder la partie entière de $|x|$: $9 = (1001)_2$
3. coder la partie fractionnaire de $|x|$: $0.75 = (0.11)_2$
4. et coder $|x|$ en notation scientifique normalisée : $m \in [1, 2[$
 $(1001)_2 + (0.11)_2 = (1001.11)_2 = (1.00111)_2 \cdot 2^3$
 $= (1.00111)_2 \cdot 2^{(11)_2}$

Cette démarche en 4 étapes conduit au résultat $x = (-1)^s \cdot m \cdot b^p = (-1)^1 \cdot (1.00111)_2 \cdot 2^{(11)_2}$ où $b = 2$, $s = (1)_2$, $m = (1.00111)_2$ et $p = (11)_2$.

Déterminer le signe, la mantisse et l'exposant binaires du nombre fractionnaire $x = 140.8125$ en suivant les 4 étapes décrites ci-dessus.

TD 3.3 : DÉCODAGE BASE B \rightarrow DÉCIMAL

La valeur décimale d'un nombre entier codé en base b peut être obtenue par l'algorithme suivant :

```
>>> n = 0
>>> for i in range(len(code)) :
...   n = n + code[i]*b**(len(code)-1-i)
...
>>>
```

Spécifier une fonction qui encapsulera cet algorithme.

TD 3.4 : CODAGE DES ENTIERS POSITIFS (2)

Définir (spécifier et implémenter) une fonction qui code un entier n en base b sur k chiffres (voir TD 3.1).

TD 3.5 : UNE SPÉCIFICATION, DES IMPLÉMENTATIONS

1. Proposer deux implémentations du calcul de la somme $s = \sum_0^n u_k$ des n premiers termes d'une suite géométrique $u_k = a \cdot b^k$.
2. Comparer les complexités de ces deux implémentations.

TD 3.6 : PASSAGE PAR VALEUR

On considère les codes suivants :

<pre>>>> x, y (1, 2) >>> tmp = x >>> x = y >>> y = tmp >>> x, y (2, 1)</pre>	<pre>def swap(x,y) : tmp = x x = y y = tmp return</pre>	<pre>>>> x, y (1, 2) >>> swap(x,y) >>> x, y (1, 2)</pre>
---	---	---

Expliquer la différence entre l'exécution de gauche et l'exécution de droite en explicitant l'appel équivalent à l'appel `swap(x,y)` dans l'exécution de droite.

TD 3.7 : VALEURS PAR DÉFAUT

On considère la fonction qui code en base b sur k chiffres un nombre décimal n (voir TD 3.4). Proposer une définition de cette fonction où $k = 8$ et $b = 2$ par défaut.

TD 3.8 : PORTÉE DES VARIABLES

On considère les fonctions `f`, `g` et `h` suivantes :

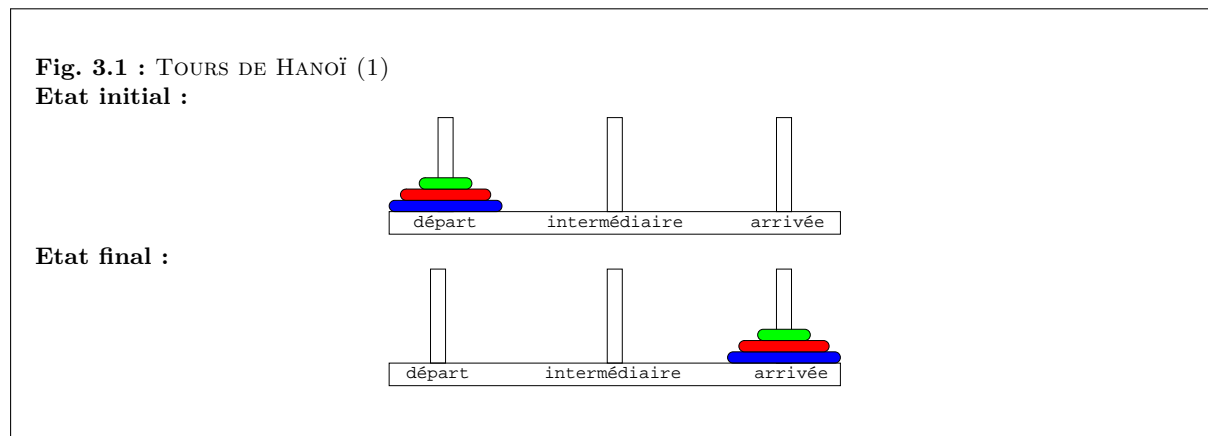
<pre>def f(x) : x = 2*x print('f', x) return x</pre>	<pre>def g(x) : x = 2*f(x) print('g', x) return x</pre>	<pre>def h(x) : x = 2*g(f(x)) print('h', x) return x</pre>
--	---	--

Qu'affichent les appels suivants ?

- | | |
|---|---|
| <p>1. >>> x = 5
 >>> print(x)
 ?
 >>> y = f(x)
 >>> print(x)
 ?
 >>> z = g(x)
 >>> print(x)
 ?
 >>> t = h(x)
 >>> print(x)
 ?</p> | <p>2. >>> x = 5
 >>> print(x)
 ?
 >>> x = f(x)
 >>> print(x)
 ?
 >>> x = g(x)
 >>> print(x)
 ?
 >>> x = h(x)
 >>> print(x)
 ?</p> |
|---|---|

TD 3.9 : TOURS DE HANOÏ à la main

Résoudre à la main le problème des tours de Hanoï à n disques (voir figure 3.1) successivement pour $n = 1$, $n = 2$, $n = 3$ et $n = 4$.



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

1. Définir une fonction récursive qui calcule le plus grand commun diviseur d de 2 entiers a et b :
 $\text{pgcd}(a, b) = \text{pgcd}(b, a \bmod b) = \dots = \text{pgcd}(d, 0) = d$.
2. En déduire une fonction qui calcule le plus petit commun multiple m de 2 entiers a et b .

TD 3.11 : SOMME ARITHMÉTIQUE

1. Définir une fonction récursive qui calcule la somme des n premiers nombres entiers.

$$s = \sum_{k=0}^n k = \frac{n(n+1)}{2}$$

2. Comparer la complexité de cette version avec les versions constante et itérative (voir TD 3.5).

TD 3.12 : COURBES FRACTALES

On s'intéresse ici aux programmes dont l'exécution produit des dessins à l'aide de la tortue Logo. On considère la procédure `draw` ci-dessous :

```
def draw(n,d) :
    assert type(n) is int
    assert n >= 0
    if n == 0 : forward(d)
    else :
        draw(n-1,d/3.)
        left(60)
        draw(n-1,d/3.)
        right(120)
        draw(n-1,d/3.)
        left(60)
        draw(n-1,d/3.)
    return
```

Dessiner le résultat des appels `draw(n,900)` respectivement pour $n = 0$, $n = 1$, $n = 2$ et $n = 3$. A chaque appel, le crayon est initialement en $(0,0)$ avec une direction de 0.

TD 3.13 : QCM (3)*(un seul item correct par question)*

1. La réutilisabilité d'un algorithme est son aptitude
 - (a) à utiliser de manière optimale les ressources du matériel qui l'exécute
 - (b) à se protéger de conditions anormales d'utilisation
 - (c) à résoudre des tâches équivalentes à celle pour laquelle il a été conçu
 - (d) à réaliser exactement la tâche pour laquelle il a été conçu
2. L'encapsulation est l'action
 - (a) de mettre une chose dans une autre
 - (b) de fermer une chose par une autre
 - (c) de substituer une chose par une autre
 - (d) de remplacer une chose par une autre
3. Une fonction est un bloc d'instructions nommé et paramétré
 - (a) qui ne peut pas retourner plusieurs valeurs
 - (b) qui ne peut pas contenir d'instructions itératives
 - (c) qui retourne une valeur
 - (d) qui ne retourne pas de valeur
4. Les paramètres d'entrée d'une fonction sont
 - (a) les arguments nécessaires pour effectuer le traitement associé à la fonction
 - (b) les valeurs obtenues après avoir effectué le traitement associé à la fonction
 - (c) des grandeurs invariantes pendant l'exécution de la fonction
 - (d) des variables auxiliaires définies dans le corps de la fonction
5. Les préconditions d'une fonction sont des conditions à respecter
 - (a) par les paramètres de sortie de la fonction
 - (b) pendant toute l'exécution de la fonction
 - (c) par les paramètres d'entrée de la fonction
 - (d) pour pouvoir compiler la fonction
6. La description d'une fonction décrit
 - (a) ce que fait la fonction
 - (b) comment fait la fonction
 - (c) pourquoi la fonction le fait
 - (d) où la fonction le fait
7. Le jeu de tests d'une fonction est
 - (a) un ensemble d'exercices à résoudre
 - (b) un ensemble d'exceptions dans le fonctionnement de la fonction
 - (c) un ensemble caractéristiques d'entrées-sorties associées
 - (d) un ensemble de recommandations dans l'utilisation de la fonction
8. En PYTHON, l'instruction **assert** permet de
 - (a) tester une précondition

- (b) imposer une instruction
 - (c) paramétrer une fonction
 - (d) tester un test du jeu de tests
9. La validité d'une fonction est son aptitude à réaliser exactement la tâche pour laquelle elle a été conçue. Plus concrètement,
- (a) la fonction doit vérifier impérativement ses préconditions
 - (b) la fonction doit être correctement paramétrée
 - (c) l'implémentation de la fonction doit être conforme aux jeux de tests
 - (d) l'utilisation de la fonction doit être conviviale
10. Le passage des paramètres par valeur consiste à copier
- (a) la valeur du paramètre formel dans le paramètre effectif correspondant
 - (b) la référence du paramètre effectif dans le paramètre formel correspondant
 - (c) la référence du paramètre formel dans le paramètre effectif correspondant
 - (d) la valeur du paramètre effectif dans le paramètre formel correspondant
11. Un appel récursif est un appel
- (a) dont l'exécution est un processus récursif
 - (b) dont l'exécution est un processus itératif
 - (c) dont le résultat est retourné par la fonction
 - (d) d'une fonction par elle-même

TD 3.14 : PASSAGE DES PARAMÈTRES

On considère les fonctions **f**, **g** et **h** suivantes :

```
def f(x):
    y = x + 2
    return y
```

```
def g(z):
    v = 2*f(z)
    return v
```

```
def h(a):
    b = g(f(a))
    return b
```

Quels sont les algorithmes équivalents (algorithmes où il n'y a plus d'appels aux fonctions **f**, **g** et **h**) aux appels suivants :

1. **u** = **f**(2)

3. **u** = **g**(2)

5. **u** = **h**(2)

2. **u** = **f**(**t**)

4. **u** = **g**(**t**)

6. **u** = **h**(**t**)

TD 3.15 : PORTÉE DES VARIABLES (2)

On considère les fonctions **f**, **g** et **h** suivantes :

```
def f(x):
    x = x + 2
    print('f', x)
    return x
```

```
def g(x):
    x = 2*f(x)
    print('g', x)
    return x
```

```
def h(x):
    x = g(f(x))
    print('h', x)
    return x
```

Qu'affichent les appels suivants ?

- | | |
|--|---|
| <p>1. <code>>>> x = 5</code>
 <code>>>> print(x)</code></p> <p><code>>>> x = x + 2</code>
 <code>>>> print(x)</code></p> <p><code>>>> x = 2 * (x + 2)</code>
 <code>>>> print(x)</code></p> | <p>4. <code>>>> x = 5</code>
 <code>>>> print(x)</code></p> <p><code>>>> f(x)</code>
 <code>>>> print(x)</code></p> <p><code>>>> g(x)</code>
 <code>>>> print(x)</code></p> |
| <p>2. <code>>>> x = 5</code>
 <code>>>> print(x)</code></p> <p><code>>>> y = f(x)</code>
 <code>>>> print(x, y)</code></p> <p><code>>>> z = 2*f(y)</code>
 <code>>>> print(x, y, z)</code></p> | <p><code>>>> h(x)</code>
 <code>>>> print(x)</code></p> |
| <p>3. <code>>>> x = 5</code>
 <code>>>> print(x)</code></p> <p><code>>>> z = 2*f(f(x))</code>
 <code>>>> print(x, z)</code></p> | |

TD 3.16 : SUITE GÉOMÉTRIQUE

Définir une fonction récursive qui calcule la somme des n premiers termes d'une suite géométrique $u_k = ab^k$.

TD 3.17 : PUISSANCE ENTIÈRE

Définir une fonction récursive qui calcule la puissance entière $p = x^n$ d'un nombre entier x .

TD 3.18 : COEFFICIENTS DU BINÔME

Définir une fonction récursive qui calcule les coefficients du binôme :

$$(a + b)^n = \sum_{k=0}^n \frac{n!}{k!(n-k)!} a^{n-k} b^k.$$

TD 3.19 : FONCTION D'ACKERMAN

Définir une fonction récursive qui calcule la fonction d'Ackerman :

$$f : N^2 \rightarrow N \quad \begin{cases} f(0, n) &= n + 1 \\ f(m, 0) &= f(m - 1, 1) \text{ si } m > 0 \\ f(m, n) &= f(m - 1, f(m, n - 1)) \text{ si } m > 0, n > 0 \end{cases}$$

TD 3.20 : ADDITION BINAIRE

Définir une fonction `add2` qui effectue l'addition binaire de 2 entiers a et b (le nombre de bits n'est pas limité *a priori*).

Exemple : $(0101)_2 + (10011)_2 = (11000)_2$

```
# add2(a,b)
>>> add2([1,0],[1,0,1,1])
[1, 1, 0, 1]
>>> add2([1,0,1,1],[1,0])
[1, 1, 0, 1]
>>> add2([1,1],[1,1])
[1, 1, 0]
```

TD 3.21 : COMPLÉMENT À 2

Définir une fonction `neg2` qui détermine le complément à 2 en binaire d'un entier n codé sur k bits.

$$(011100)_2 \rightarrow (100100)_2 : \begin{array}{r} (011100)_2 \\ (100011)_2 \\ + (000001)_2 \\ \hline = (100100)_2 \end{array}$$

```
# neg2(code)
>>> neg2([0,0,0,1,0,1,1,1])
[1, 1, 1, 0, 1, 0, 0, 1]
>>> neg2([1, 1, 1, 0, 1, 0, 0, 1])
[0, 0, 0, 1, 0, 1, 1, 1]
>>> for a in [0,1]:
...     for b in [0,1]:
...         for c in [0,1]:
...             add2([a,b,c],neg2([a,b,c]))
[0, 0, 0, 0]
[1, 0, 0, 0]
[1, 0, 0, 0]
[1, 0, 0, 0]
[1, 0, 0, 0]
[1, 0, 0, 0]
[1, 0, 0, 0]
[1, 0, 0, 0]
```

TD 3.22 : CODAGE-DÉCODAGE DES RÉELS

1. Définir une fonction `ieee` qui code un nombre réel x selon la norme IEEE 754 simple précision.

$$x = (-1)^s \cdot (1 + m) \cdot 2^{(e-127)}$$

```
# ieee(x)
>>> ieee(0.0)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> ieee(0.625)
[0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> ieee(3.1399998664855957)
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0]
>>> ieee(-4573.5)
[1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0,
 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

2. Définir une fonction `real` qui décode un nombre réel x codé selon la norme IEEE 754 simple précision.

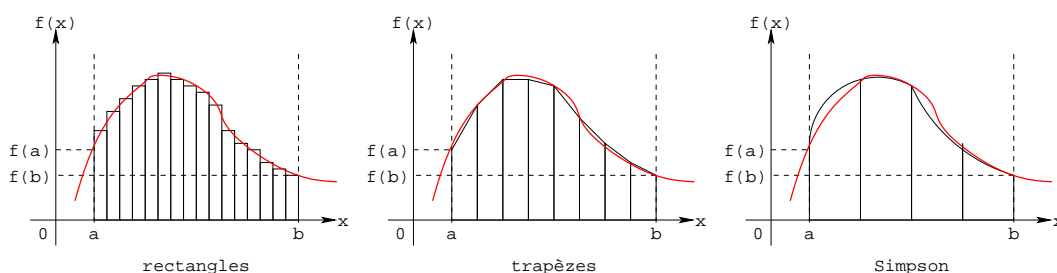
$$x = (-1)^s \cdot (1 + m) \cdot 2^{(e-127)}$$

```
# real(code)
>>> real(ieee(0.625))
0.625
>>> real(ieee(3.1399998664855957))
3.1399998664855957
>>> real(ieee(-4573.5))
-4573.5
```

On pourra vérifier les résultats obtenus avec la fonction `ieee` du TD 3.22 ci-contre sur le site <http://babbage.cs.qc.edu/IEEE-754/Decimal.html>

TD 3.23 : INTÉGRATION NUMÉRIQUE

Soit $f(x)$ une fonction continue de $\mathbb{R} \rightarrow \mathbb{R}$ à intégrer sur $[a, b]$ (on supposera que f a toutes les bonnes propriétés mathématiques pour être intégrable sur l'intervalle considéré). On cherche à calculer son intégrale $I = \int_a^b f(x)dx$ qui représente classiquement l'aire comprise entre la courbe représentative de f et les droites d'équations $x = a$, $x = b$ et $y = 0$. Les méthodes d'intégration numérique (méthode des rectangles, méthode des trapèzes et méthode de Simpson) consistent essentiellement à trouver une bonne approximation de cette aire.



On testera ces différentes méthodes avec la fonction $f(x) = \sin(x)$ sur $[0, \pi]$.

1. Méthode des rectangles : subdivisons l'intervalle d'intégration de longueur $b-a$ en n parties égales de longueur $\Delta x = \frac{b-a}{n}$. Soient x_1, x_2, \dots, x_n les points milieux de ces n intervalles. Les n rectangles formés avec les ordonnées correspondantes ont pour surface $f(x_1)\Delta x$, $f(x_2)\Delta x$, ..., $f(x_n)\Delta x$. L'aire sous la courbe est alors assimilée à la somme des aires de ces rectangles, soit

$$I = \int_a^b f(x)dx \approx (f(x_1) + f(x_2) + \dots + f(x_n)) \Delta x$$

C'est la formule dite des rectangles qui repose sur une approximation par une fonction *en escalier*.

Ecrire une fonction `rectangle_integration` qui calcule l'intégrale définie I d'une fonction f sur $[a, b]$ à l'ordre n par la méthode des rectangles.

2. Méthode des trapèzes : subdivisons l'intervalle d'intégration de longueur $b-a$ en n parties égales de longueur $\Delta x = \frac{b-a}{n}$. Les abscisses des points ainsi définis sont $a, x_1, x_2, \dots, x_{n-1}, b$ et les trapèzes construits sur ces points et les ordonnées correspondantes ont pour aire $\frac{\Delta x}{2} (f(a) + f(x_1))$, $\frac{\Delta x}{2} (f(x_1) + f(x_2))$, ..., $\frac{\Delta x}{2} (f(x_{n-1}) + f(b))$. L'aire sous la courbe est alors assimilée à la somme des aires de ces trapèzes, soit

$$I = \int_a^b f(x)dx \approx \left(\frac{f(a) + f(b)}{2} + f(x_1) + f(x_2) + \dots + f(x_{n-1}) \right) \Delta x$$

C'est la formule dite des trapèzes.

Ecrire une fonction `trapezoid_integration` qui calcule l'intégrale définie I d'une fonction f sur $[a, b]$ à l'ordre n par la méthode des trapèzes.

3. Méthode de Simpson : divisons l'intervalle d'intégration $[a, b]$ en un nombre n pair d'intervalles dont la longueur est $\Delta x = \frac{b-a}{n}$. Dans les 2 premiers intervalles d'extrémités a , x_1 et x_2 , on approche la courbe représentative de f par une parabole d'équation $y = \alpha x^2 + \beta x + \gamma$ passant par les points $A(a, f(a))$, $A_1(x_1, f(x_1))$ et $A_2(x_2, f(x_2))$ de la courbe. Dans les 2 intervalles suivants, on approche la courbe par une autre parabole d'équation similaire, passant par les points A_2 , A_3 et A_4 , et ainsi de suite. On obtient ainsi une courbe formée de n portions de parabole et l'aire déterminée par ces portions de parabole est une approximation de l'aire I cherchée.

L'intégration de l'équation de la parabole $y = \alpha x^2 + \beta x + \gamma$ sur $[-\Delta x, \Delta x]$ donne

$$S = \int_{-\Delta x}^{\Delta x} (\alpha x^2 + \beta x + \gamma) dx = \frac{2}{3} \alpha (\Delta x)^3 + 2\gamma (\Delta x)$$

où les constantes α et γ sont déterminées en écrivant que les points $(-\Delta x, y_0)$, $(0, y_1)$ et $(\Delta x, y_2)$ satisfont l'équation de la parabole. On obtient ainsi :

$$\begin{cases} y_0 = \alpha(-\Delta x)^2 + \beta(-\Delta x) + \gamma \\ y_1 = \gamma \\ y_2 = \alpha(\Delta x)^2 + \beta(\Delta x) + \gamma \end{cases} \Rightarrow \begin{cases} \alpha = \frac{y_0 - 2y_1 + y_2}{2(\Delta x)^2} \\ \beta = \frac{y_2 - y_0}{2(\Delta x)} \\ \gamma = y_1 \end{cases}$$

et $S = \frac{\Delta x}{3}(y_0 + 4y_1 + y_2)$.

$$\text{Par suite, il vient : } \begin{cases} S_1 = \frac{\Delta x}{3}(y_0 + 4y_1 + y_2) \\ S_2 = \frac{\Delta x}{3}(y_2 + 4y_3 + y_4) \\ S_3 = \frac{\Delta x}{3}(y_4 + 4y_5 + y_6) \\ \vdots = \\ S_{n/2} = \frac{\Delta x}{3}(y_{n-2} + 4y_{n-1} + y_n) \end{cases}$$

d'où

$$I = \int_a^b f(x) dx \approx \frac{\Delta x}{3} \left(f(a) + 4 \sum_{i=1,3,5,\dots}^{n-1} f(x_i) + 2 \sum_{i=2,4,6,\dots}^{n-2} f(x_i) + f(b) \right)$$

C'est la formule dite de Simpson qui repose sur une approximation de f par des arcs de parabole.

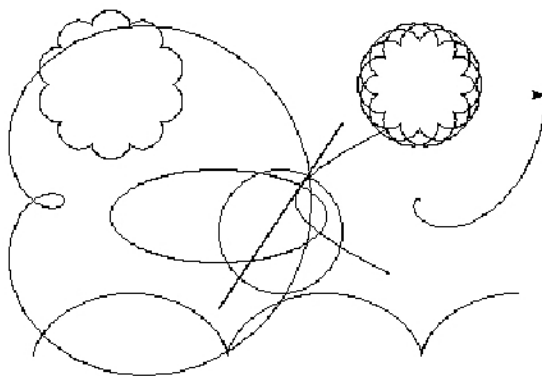
Ecrire une fonction `simpson_integration` qui calcule l'intégrale définie I d'une fonction f sur $[a, b]$ à l'ordre n par la méthode de Simpson.

TD 3.24 : TRACÉS DE COURBES PARAMÉTRÉES

Une courbe paramétrée dans le plan est une courbe où l'abscisse x et l'ordonnée y sont des fonctions d'un paramètre qui peut être le temps t ou un angle θ par exemple. La courbe se présente donc sous la forme $x = f(t)$, $y = g(t)$. Les tableaux ci-dessous en donnent quelques exemples.

droite	$x = x_0 + \alpha t$ $y = y_0 + \beta t$	cycloïde	$x = x_0 + r(\phi - \sin(\phi))$ $y = y_0 + r(1 - \cos(\phi))$
cercle	$x = x_0 + r \cos(\theta)$ $y = y_0 + r \sin(\theta)$	épicycloïde	$x = x_0 + (R + r) \cos(\theta) - r \cos\left(\frac{R+r}{r} \cdot \theta\right)$ $y = y_0 + (R + r) \sin(\theta) - r \sin\left(\frac{R+r}{r} \cdot \theta\right)$
ellipse	$x = x_0 + a \cos(\phi)$ $y = y_0 + b \cos(\phi)$	hypercycloïde	$x = x_0 + (R - r) \cos(\theta) + r \cos\left(\frac{R-r}{r} \cdot \theta\right)$ $y = y_0 + (R - r) \sin(\theta) + r \sin\left(\frac{R-r}{r} \cdot \theta\right)$
hyperbole	$x = x_0 + \frac{a}{\cos(\theta)}$ $y = y_0 + b \tan(\theta)$		

limaçon de Pascal	$x = x_0 + (a \cos(\theta) + b) \cos(\theta)$ $y = y_0 + (a \cos(\theta) + b) \sin(\theta)$
spirale logarithmique	$x = x_0 + k e^\theta \cos(\theta)$ $y = y_0 + k e^\theta \sin(\theta)$

Fig. 3.2 : COURBES PARAMÉTRÉES

Ecrire une fonction **drawCurve** qui permettent le tracé de telles courbes paramétrées (figure 3.2). On utilisera les instructions *à la* LOGO pour réaliser ces tracés (voir annexe 5.5 page 51).

Chapitre 4

Structures linéaires

TD 4.1 : DISTANCE DE 2 POINTS DE L'ESPACE

Définir la fonction `distance` qui calcule la distance entre 2 points M_1 et M_2 de l'espace, respectivement de coordonnées (x_1, y_1, z_1) et (x_2, y_2, z_2) .

TD 4.2 : OPÉRATIONS SUR LES N-UPLETS

Donner un exemple d'utilisation de chacune des opérations sur les n-uplets décrites dans le tableau de la page 52.

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

(voir TD 3.10)

Définir une fonction qui calcule le pgcd et le ppcm de 2 entiers a et b .

TD 4.4 : OPÉRATIONS SUR LES CHÂÎNES

Donner un exemple d'utilisation de chacune des opérations sur les chaînes de caractères décrites dans le tableau de la page 52.

TD 4.5 : INVERSER UNE CHÂÎNE

Définir une fonction qui crée une copie d'une chaîne en inversant l'ordre des caractères.

```
>>> inverser('inverser')
'resrevni'
```

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

Définir une fonction qui compte le nombre de caractères, le nombre de mots et le nombres de lignes d'une chaîne de caractères.

TD 4.7 : OPÉRATIONS SUR LES LISTES (1)

Donner un exemple d'utilisation de chacune des opérations sur les listes décrites dans le tableau de la page 52.

TD 4.8 : OPÉRATIONS SUR LES LISTES (2)

Vérifier que les opérations suivantes sont équivalentes deux à deux :

```
del s[i:j]      et s[i:j] = []
s.append(x)     et s[len(s):len(s)] = [x]
s.extend(x)     et s[len(s):len(s)] = x
s.insert(i,x)   et s[i:i] = [x]
s.remove(x)     et del s[s.index(x)]
s.pop(i)        et x = s[i] ; del s[i]
```

TD 4.9 : SÉLECTION D'ÉLÉMENTS

1. Définir une fonction qui crée une liste **t** composée des éléments d'une autre liste **s** qui vérifient une certaine condition **p** ($p(s[i]) == \text{True}$).
2. Définir une fonction qui supprime d'une liste **s** les éléments qui ne vérifient pas une certaine condition **p**.

TD 4.10 : OPÉRATIONS SUR LES PILES

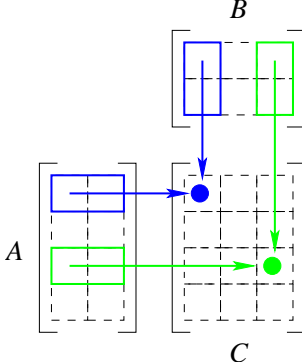
Définir les 4 opérations sur les piles définies ci-contre : **emptyStack**, **topStack**, **pushStack** et **popStack**. On empilera et dépilera à la fin de la liste qui sert à stocker les éléments de la pile.

TD 4.11 : OPÉRATIONS SUR LES FILES

Définir les 4 opérations sur les files définies ci-contre : **emptyQueue**, **topQueue**, **pushQueue** et **popQueue**. On enfilera en début de liste et on défilera à la fin de la liste qui sert à stocker les éléments de la file.

TD 4.12 : PRODUIT DE MATRICES

Définir la fonction qui calcule la matrice **C**, produit de 2 matrices **A** et **B** respectivement de dimensions (n, r) et (r, m) .

$$c_{i,j} = \sum_{k=0}^{r-1} a_{ik} \cdot b_{kj}$$


TD 4.13 : ANNUAIRE TÉLÉPHONIQUE

On considère un annuaire téléphonique stocké sous la forme d'une liste de couples (**nom**, **téléphone**) (exemple : $[('jean', '0607080910'), ('paul', '0298000102')]$).

1. Définir une fonction qui retrouve dans un annuaire téléphonique le numéro de téléphone à partir du nom.
2. Définir la fonction inverse qui retrouve le nom à partir du numéro de téléphone.

TD 4.14 : RECHERCHE DICHOTOMIQUE

La recherche dichotomique présentée ci-contre n'assure pas de trouver la première occurrence d'un élément **x** dans une liste **t** triée.

Modifier l'algorithme de recherche dichotomique proposé pour rechercher la première occurrence d'un élément **x** dans une liste **t** triée.

TD 4.15 : LISTE ORDONNÉE

Définir une version itérative de la fonction récursive **enOrdre** définie ci-dessous. On appliquera la méthode de transformation décrite en section 5.9 page 56.


```
def enOrdre(t,debut,fin):
    assert type(t) is list
    assert 0 <= debut <= fin < len(t)
    ok = False
    if debut == fin: ok = True
    else:
        if t[debut] <= t[debut+1]:
            ok = enOrdre(t,debut+1,fin)
    return ok
```

TD 4.16 : TRI D'UN ANNUAIRE TÉLÉPHONIQUE

On considère un annuaire téléphonique stocké sous la forme d'une liste de couples (nom, téléphone) (exemple : [('paul', '0607080910'), ('jean', '0298000102')]).

Définir une fonction qui trie un annuaire téléphonique par ordre alphabétique des noms.

TD 4.17 : COMPLEXITÉ DU TRI PAR SÉLECTION

Montrer que la complexité du tri par sélection est :

1. en $O(n)$ en ce qui concerne les échanges de valeurs au sein d'une liste de longueur n ,
2. en $O(n^2)$ en ce qui concerne les comparaisons entre éléments de la liste.

TD 4.18 : TRI PAR INSERTION

Exécuter à la main l'appel `triInsertion([9,8,7,6,5,4],1,4)` en donnant les valeurs des variables `i`, `k`, `x` et `t` à la fin de chaque itération de la boucle `while` (lignes 7–9 de la version itérative de l'algorithme).

Tri par insertion

```
1 def triInsertion(t,debut,fin):
2     assert type(t) is list
3     assert 0 <= debut <= fin < len(t)
4     for k in range(debut+1,fin+1):
5         i = k - 1
6         x = t[k]
7         while i >= debut and t[i] > x:
8             t[i+1] = t[i]
9             i = i - 1
10        t[i+1] = x
11    return
```

TD 4.19 : COMPARAISON D'ALGORITHMES (1)

Comparer les temps d'exécution des versions récursive et itérative du tri rapide pour différentes tailles de liste.

On pourra utiliser la fonction `random()` ou `randint(min,max)` du module standard `random` pour générer des listes de nombres aléatoires, et la fonction `time()` du module standard `time` pour mesurer le temps avant et après l'appel des fonctions.

TD 4.20 : QCM (4)

(un seul item correct par question)

1. Le type d'une variable définit
 - (a) l'ensemble des noms qu'elle peut prendre.

- (b) le regroupement fini des données qui la composent et dont le nombre n'est pas fixé *a priori*.
 - (c) l'ensemble des valeurs qu'elle peut prendre et l'ensemble des opérations qu'elle peut subir.
 - (d) le regroupement hiérarchique des données qui la composent
2. Une séquence est
- (a) un regroupement fini de données dont le nombre n'est pas fixé *a priori*.
 - (b) une collection d'éléments, appelés « sommets », et de relations entre ces sommets.
 - (c) une collection non ordonnée d'éléments.
 - (d) une suite ordonnée d'éléments accessibles par leur rang dans la séquence.
3. Parmi les exemples suivants, le seul exemple de séquence est
- (a) le tableau final d'un tournoi de tennis.
 - (b) la carte routière.
 - (c) la classification des espèces animales.
 - (d) la main au poker.
4. Parmi les types suivants, un seul n'est pas une variante de séquence. Lequel ?
- (a) le n-uplet.
 - (b) la chaîne de caractères.
 - (c) le dictionnaire.
 - (d) la pile.
5. Dans une liste
- (a) tous les éléments sont du même type.
 - (b) les éléments peuvent avoir des types différents.
 - (c) les éléments ne peuvent pas être des dictionnaires.
 - (d) les éléments ont comme type un des types de base (`bool`, `int`, `float`).
6. Dans la liste multidimensionnelle `s = [[1,2,3,4], [5,6,7], [8,9]]` que vaut `s[1][1]` ?
- (a) 1.
 - (b) 2.
 - (c) 5.
 - (d) 6.
7. Une pile est une séquence dans laquelle
- (a) on ne peut ajouter un élément qu'à une seule extrémité et ne supprimer un élément qu'à l'autre extrémité.
 - (b) on ne peut ajouter un élément qu'à une seule extrémité et en supprimer n'importe où.
 - (c) on ne peut ajouter et supprimer un élément qu'à une seule extrémité.
 - (d) on ne peut supprimer un élément qu'à une seule extrémité et en ajouter n'importe où.
8. Une file est une séquence dans laquelle

- (a) on ne peut ajouter un élément qu'à une seule extrémité et ne supprimer un élément qu'à l'autre extrémité.
 - (b) on ne peut ajouter un élément qu'à une seule extrémité et en supprimer n'importe où.
 - (c) on ne peut ajouter et supprimer un élément qu'à une seule extrémité.
 - (d) on ne peut supprimer un élément qu'à une seule extrémité et en ajouter n'importe où.
9. La recherche séquentielle d'un élément dans une liste consiste à
- (a) rechercher le minimum de la liste et à le mettre en début de liste en l'échangeant avec cet élément.
 - (b) rechercher le maximum de la liste et à le mettre en début de liste en l'échangeant avec cet élément.
 - (c) comparer l'élément recherché successivement à tous les éléments de la liste jusqu'à trouver une correspondance.
 - (d) comparer l'élément recherché avec l'élément milieu de la liste et poursuivre de même dans la sous-liste de droite ou dans la sous-liste de gauche à l'élément milieu.
10. Dans le tri par insertion
- (a) on partage la liste à trier en deux sous-listes telles que tous les éléments de la première soient inférieurs à tous les éléments de la seconde, puis on trie les deux sous-listes selon le même processus jusqu'à avoir des sous-listes réduites à un seul élément.
 - (b) on trie successivement les premiers éléments de la liste : à la $i^{\text{ème}}$ étape, on place le $i^{\text{ème}}$ élément à son rang parmi les $i - 1$ éléments précédents qui sont déjà triés entre eux.
 - (c) on parcourt la liste en commençant par la fin, en effectuant un échange à chaque fois que l'on trouve deux éléments successifs qui ne sont pas dans le bon ordre.
 - (d) on recherche le minimum de la liste à trier, on le met en début de liste en l'échangeant avec le premier élément et on recommence sur le reste de la liste.

TD 4.21 : GÉNÉRATION DE SÉQUENCES

A l'aide de la fonction `randint(min,max)` du module standard `random`, Définir les fonctions de génération suivantes :

1. `liste(n)` : génère une liste de `n` entiers compris entre 0 et `n`.
2. `nuplet(n)` : génère un `n`-uplet de `n` entiers compris entre 0 et `n`.
3. `chaîne(n)` : génère une chaîne de `n` caractères imprimables.

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

Définir une fonction qui applique la même fonction `f` à tous les éléments d'une liste `s`.

Exemples : `s = [-1,2,3,-4]` , `f = abs` \rightarrow `s = [1,2,3,4]`
`s = [pi/2,pi,3*pi/2]` , `f = sin` \rightarrow `s = [1,0,-1]`

TD 4.23 : QUE FAIT CETTE PROCÉDURE ?

On considère la procédure `f` ci-dessous.

```

def f(t,debut,fin):
    m = (debut + fin)/2
    while m > 0:
        for i in range(m,fin+1):
            j = i - m
            while j >= debut:
                print(m,i,j,t)
                if t[j] > t[j+m]:
                    t[j],t[j+m] = t[j+m],t[j]
                    j = j - m
                else: j = debut-1
            m = m/2
    return

```

1. Tracer l'exécution à la main de l'appel `f([4,2,1,2,3,5],0,5)`.
2. Que fait cette procédure?

TD 4.24 : CODES ASCII ET CHAÎNES DE CARACTÈRES

1. Ecrire un algorithme qui fournit le tableau des codes ASCII (annexe 5.1 page 47) associé à une chaîne (exemple : 'bon' → [98, 111, 110]).
2. Ecrire un algorithme qui donne la chaîne de caractères associée à un tableau de codes ASCII (exemple : [98, 111, 110] → 'bon').

TD 4.25 : OPÉRATIONS SUR LES MATRICES

1. Définir une fonction qui teste si une liste `m` est une matrice.
2. Définir une fonction qui teste si une matrice `m` est une matrice carrée.
3. Définir une fonction qui teste si une matrice `m` est une matrice symétrique.
4. Définir une fonction qui teste si une matrice `m` est une matrice diagonale.
5. Définir une fonction qui multiplie une matrice `m` par un scalaire `x`.
6. Définir une fonction qui détermine la transposée d'une matrice `m`.

TD 4.26 : RECHERCHE D'UN MOTIF

Définir un algorithme qui recherche la première occurrence d'une séquence `m` au sein d'une autre séquence `t`.

TD 4.27 : RECHERCHE DE TOUTES LES OCCURENCES

Définir une fonction qui retourne la liste des rangs de toutes les occurrences d'un élément `x` dans une liste `t`.

TD 4.28 : TRI BULLES

Dans le tri bulles, on parcourt la liste en commençant par la fin, en effectuant un échange à chaque fois que l'on trouve deux éléments successifs qui ne sont pas dans le bon ordre.

Définir une fonction qui trie une liste selon la méthode du tri bulles.

TD 4.29 : MÉTHODE D'ÉLIMINATION DE GAUSS

L'objectif est ici de résoudre dans \mathbb{R} un système de n équations linéaires à n inconnues, homogènes ou non homogènes, du type $A \cdot x = b$:

$$\left\{ \begin{array}{l} a_{00}x_0 + a_{01}x_1 + \cdots + a_{0(n-1)}x_{(n-1)} = b_0 \\ a_{10}x_0 + a_{11}x_1 + \cdots + a_{1(n-1)}x_{(n-1)} = b_1 \\ \cdots + \cdots + \cdots + \cdots = \cdots \\ a_{(n-1)0}x_0 + a_{(n-1)1}x_1 + \cdots + a_{(n-1)(n-1)}x_{(n-1)} = b_{(n-1)} \end{array} \right.$$

Définir une fonction `solve(a,b)` qui retourne le vecteur `x` solution du système linéaire $A \cdot x = b$ selon la méthode d'élimination de GAUSS décrite en section 5.10 page 57.

TD 4.30 : COMPARAISON D'ALGORITHMES DE RECHERCHE.

Comparer les temps d'exécution des versions récursive et itérative des 2 méthodes de recherche développées dans le cours (recherche séquentielle, recherche dichotomique).

On pourra utiliser la fonction `random()` ou `randint(min,max)` du module standard `random` pour générer des listes de nombres aléatoires, et la fonction `time()` du module standard `time` pour mesurer le temps avant et après l'appel des fonctions.

TD 4.31 : COMPARAISON D'ALGORITHMES DE TRI

Comparer les temps d'exécution des différentes versions récursives et itératives des 3 méthodes de tri développées dans le cours (tri par sélection, tri par insertion, tri rapide).

On pourra utiliser la fonction `random()` ou `randint(min,max)` du module standard `random` pour générer des listes de nombres aléatoires, et la fonction `time()` du module standard `time` pour mesurer le temps avant et après l'appel des fonctions.

Chapitre 5

Annexes

5.1 Codes ASCII

L'ordinateur stocke toutes les données sous forme numérique (ensemble de bits). En particulier, les caractères ont un équivalent numérique : le code ASCII (*American Standard Code for Information Interchange*). Le code ASCII de base code les caractères sur 7 bits (128 caractères, de 0 à 127).

- Les codes 0 à 31 sont des caractères de contrôle (figure 5.1).
- Les codes 32 à 47, de 58 à 64, de 91 à 96 et de 123 à 126 sont des symboles de ponctuation.

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	()	*	+	,	-	.	/

58	59	60	61	62	63	64	91	92	93	94	95	96	123	124	125	126
:	;	<	=	>	?	@	[\]	^	_	`	{		}	~

- Les codes de 48 à 57 représentent les 10 chiffres de 0 à 9.
- Les codes 65 à 90 représentent les majuscules de A à Z,
- Les codes 97 à 122 représentent les minuscules de a à z (il suffit d'ajouter 32 au code ASCII d'une majuscule pour obtenir la minuscule correspondante).
- Les caractères accentués (é, è ...) font l'objet d'un code ASCII étendu de 128 à 255.

Fig. 5.1 : CODES ASCII DES CARACTÈRES DE CONTRÔLE

code	caractère	signification	code	caractère	signification
0	NUL	<i>Null</i>	16	DLE	<i>Data link escape</i>
1	SOH	<i>Start of heading</i>	17	DC1	<i>Device control 1</i>
2	STX	<i>Start of text</i>	18	DC2	<i>Device control 2</i>
3	ETX	<i>End of text</i>	19	DC3	<i>Device control 3</i>
4	EOT	<i>End of transmission</i>	20	DC4	<i>Device control 4</i>
5	ENQ	<i>Enquiry</i>	21	NAK	<i>Negative acknowledgement</i>
6	ACK	<i>Acknowledge</i>	22	SYN	<i>Synchronous idle</i>
7	BEL	<i>Bell</i>	23	ETB	<i>End of transmission block</i>
8	BS	<i>Backspace</i>	24	CAN	<i>Cancel</i>
9	TAB	<i>Horizontal tabulation</i>	25	EM	<i>End of medium</i>
10	LF	<i>Line Feed</i>	26	SUB	<i>Substitute</i>
11	VT	<i>Vertical tabulation</i>	27	ESC	<i>Escape</i>
12	FF	<i>Form feed</i>	28	FS	<i>File separator</i>
13	CR	<i>Carriage return</i>	29	GS	<i>Group separator</i>
14	SO	<i>Shift out</i>	30	RS	<i>Record separator</i>
15	SI	<i>Shift in</i>	31	US	<i>Unit separator</i>

5.2 Instructions PYTHON

Les principales instructions PYTHON sont listées dans les tableaux ci-dessous.

Miscellaneous statements

Statement	Result
<code>pass</code>	Null statement
<code>del name[, name]*</code>	Unbind <code>name(s)</code> from object
<code>print([s1 [, s2]*)</code>	Writes to <code>sys.stdout</code> , or to <code>fileobject</code> if supplied. Puts spaces between arguments <code>si</code> . Puts newline at end unless arguments end with <code>end=</code> (ie : <code>end=' '</code>). <code>print</code> is not required when running interactively, simply typing an expression will print its value, unless the value is <code>None</code> .
<code>input([prompt])</code>	Prints <code>prompt</code> if given. Reads input and evaluates it.

Assignment operators

Operator	Result
<code>a = b</code>	Basic assignment - assign object <code>b</code> to label <code>a</code>
<code>a += b</code>	<code>a = a + b</code>
<code>a -= b</code>	<code>a = a - b</code>
<code>a *= b</code>	<code>a = a * b</code>
<code>a /= b</code>	<code>a = a / b</code>
<code>a //= b</code>	<code>a = a // b</code>
<code>a %= b</code>	to <code>a = a % b</code>
<code>a **= b</code>	to <code>a = a ** b</code>
<code>a &= b</code>	<code>a = a & b</code>
<code>a = b</code>	<code>a = a b</code>
<code>a ^= b</code>	<code>a = a ^ b</code>
<code>a >>= b</code>	<code>a = a >> b</code>
<code>a <<= b</code>	<code>a = a << b</code>

Control flow statements

Statement	Result
<code>if condition :</code> suite [<code>elif condition : suite</code>]* [<code>else:</code> suite]	Usual if/else if/else statement.
<code>while condition :</code> suite [<code>else:</code> suite]	Usual <code>while</code> statement. The <code>else suite</code> is executed after loop exits, unless the loop is exited with <code>break</code> .
<code>for element in sequence :</code> suite [<code>else:</code> suite]	Iterates over <code>sequence</code> , assigning each element to <code>element</code> . Use built-in <code>range</code> function to iterate a number of times. The <code>else suite</code> is executed at end unless loop exited with <code>break</code> .
<code>break</code>	Immediately exits for or while loop.
<code>continue</code>	Immediately does next iteration of for or while loop.

<code>return [result]</code>	Exits from function (or method) and returns result (use a tuple to return more than one value).
------------------------------	--

Name space statements

Imported module files must be located in a directory listed in the `path` (`sys.path`).

Packages : a package is a name space which maps to a directory including module(s) and the special initialization module `__init__.py` (possibly empty).

Packages/directories can be nested. You address a module's symbol via `[package.[package...].module.symbol]`.

Statement	Result
<code>import module1 [as name1] [, module2]*</code>	Imports modules. Members of module must be referred to by qualifying with <code>[package.]module</code> name, e.g. : <pre>import sys; print(sys.argv) import package1.subpackage.module package1.subpackage.module.foo() module1 renamed as name1, if supplied.</pre>
<code>from module import name1 [as othername1] [, name2]*</code>	Imports names from module <code>module</code> in current namespace, e.g. : <pre>from sys import argv; print(argv) from package1 import module; module.foo() from package1.module import foo; foo() name1 renamed as othername1, if supplied.</pre>
<code>from module import *</code>	Imports all names in module, except those starting with <code>_</code> .

5.3 Fonctions en PYTHON

Le principe de la définition d'une fonction en PYTHON est présentée ci-dessous.

<code>def funcName([paramList]) :</code> <code> block</code>	Creates a function object and binds it to name funcName . <code>paramList := [param [, param]*]</code> <code>param := value id=value *id **id</code>
--	---

- Arguments are passed by value, so only arguments representing a mutable object can be modified (are inout parameters).
- Use **return** to return **None** from the function, or **return value** to return a value. Use a tuple to return more than one value, e.g. **return 1,2,3**.
- Keyword arguments **arg=value** specify a default value (evaluated at function definition time). They can only appear last in the param list, e.g. `foo(x, y=1, s='')`.
- Pseudo-arg ***args** captures a tuple of all remaining non-keyword args passed to the function, e.g. if `def foo(x, *args) : ...` is called `foo(1, 2, 3)`, then **args** will contain `(2,3)`.
- Pseudo-arg ****kwargs** captures a dictionary of all extra keyword arguments, e.g. if `def foo(x, **kwargs) : ...` is called `foo(1, y=2, z=3)`, then **kwargs** will contain `{'y':2,`

'z' :3}. if `def foo(x, *args, **kwargs) : ...` is called `foo(1, 2, 3, y=4, z=5)`, then `args` will contain `(2, 3)`, and `kwargs` will contain `{'y':4, 'z' :5}`.

- `args` and `kwargs` are conventional names, but other names may be used as well.
- `*args` and `**kwargs` can be "forwarded" (individually or together) to another function, e.g. `def f1(x, *args, **kwargs) : f2(*args, **kwargs)`.

5.4 Fonctions PYTHON prédéfinies

Les principales fonctions prédéfinies en PYTHON sont listées dans les tableaux ci-dessous.

Function	Result
<code>abs(x)</code>	Returns the absolute value of the number <code>x</code> .
<code>all(iterable)</code>	Returns <code>True</code> if <code>bool(x)</code> is <code>True</code> for all values <code>x</code> in the <code>iterable</code> .
<code>any(iterable)</code>	Returns <code>True</code> if <code>bool(x)</code> is <code>True</code> for any values <code>x</code> in the <code>iterable</code> .
<code>bool([x])</code>	Converts a value to a Boolean, using the standard truth testing procedure. If <code>x</code> is false or omitted, returns <code>False</code> ; otherwise returns <code>True</code> .
<code>chr(i)</code>	Returns one-character string whose ASCII code is integer <code>i</code> .
<code>cmp(x,y)</code>	Returns negative, 0, positive if <code>x <, ==, ></code> to <code>y</code> respectively.
<code>complex(real[, image])</code>	Creates a complex object (can also be done using <code>J</code> or <code>j</code> suffix, e.g. <code>1+3J</code>).
<code>dict([mapping-or-sequence])</code>	Returns a new dictionary initialized from the optional argument (or an empty dictionary if no argument). Argument may be a sequence (or anything iterable) of pairs (key,value).
<code>dir([object])</code>	Without args, returns the list of names in the current local symbol table. With a module, class or class instance <code>object</code> as arg, returns the list of names in its attr. dictionary.
<code>divmod(a,b)</code>	Returns tuple <code>(a//b, a%b)</code> .
<code>enumerate(iterable)</code>	Iterator returning pairs (index, value) of <code>iterable</code> , e.g. <code>List(enumerate('Py')) → [(0, 'P'), (1, 'y')]</code> .
<code>eval(s[, globals[, locals]])</code>	Evaluates string <code>s</code> , representing a single python expression, in (optional) <code>globals</code> , <code>locals</code> contexts. Example : <code>x = 1; assert eval('x + 1') == 2</code>
<code>execfile(file[, globals[, locals]])</code>	Executes a <code>file</code> without creating a new module, unlike <code>import</code> .
<code>filter(function, sequence)</code>	Constructs a list from those elements of <code>sequence</code> for which <code>function</code> returns true. <code>function</code> takes one parameter.
<code>float(x)</code>	Converts a number or a string to floating point.
<code>globals()</code>	Returns a dictionary containing the current global variables.
<code>help([object])</code>	Invokes the built-in help system. No argument → interactive help; if <code>object</code> is a string (name of a module, function, class, method, keyword, or documentation topic), a help page is printed on the console; otherwise a help page on <code>object</code> is generated.
<code>hex(x)</code>	Converts a number <code>x</code> to a hexadecimal string.
<code>id(object)</code>	Returns a unique integer identifier for <code>object</code> .
<code>input([prompt])</code>	Prints <code>prompt</code> if given. Reads input and evaluates it.
<code>int(x[, base])</code>	Converts a number or a string to a plain integer. Optional <code>base</code> parameter specifies base from which to convert string values.
<code>len(obj)</code>	Returns the length (the number of items) of an object (sequence, dictionary).
<code>list([seq])</code>	Creates an empty list or a list with same elements as <code>seq</code> . <code>seq</code> may be a sequence, a container that supports iteration, or an iterator object. If <code>seq</code> is already a list, returns a copy of it.
<code>locals()</code>	Returns a dictionary containing current local variables.
<code>map(function, sequence)</code>	Returns a list of the results of applying <code>function</code> to each item from <code>sequence(s)</code> .
<code>oct(x)</code>	Converts a number to an octal string.

<code>open(filename[,mode='r',[bufsize]])</code>	Returns a new file object. filename is the file name to be opened. mode indicates how the file is to be opened ('r', 'w', 'a', '+', 'b', 'U'). bufsize is 0 for unbuffered, 1 for line buffered, negative or omitted for system default, >1 for a buffer of (about) the given size.
<code>ord(c)</code>	Returns integer ASCII value of c (a string of len 1).
<code>range([start,] end [, step])</code>	Returns list of ints from \geq start and $<$ end . With 1 arg, list from 0.. arg -1. With 2 args, list from start .. end -1. With 3 args, list from start up to end by step .
<code>raw_input([prompt])</code>	Prints prompt if given, then reads string from std input (no trailing n).
<code>reload(module)</code>	Re-parses and re-initializes an already imported module .
<code>repr(object)</code>	Returns a string containing a printable and if possible evaluable representation of an object . \equiv <code>`object`</code> (using backquotes).
<code>round(x, n=0)</code>	Returns the floating point value x rounded to n digits after the decimal point.
<code>str(object)</code>	Returns a string containing a nicely printable representation of an object .
<code>sum(iterable[, start=0])</code>	Returns the sum of a sequence of numbers (not strings), plus the value of parameter. Returns start when the sequence is empty.
<code>tuple([seq])</code>	Creates an empty tuple or a tuple with same elements as seq .
<code>type(obj)</code>	Returns a type object representing the type of obj .
<code>xrange(start [, end [, step]])</code>	Like <code>range()</code> , but doesn't actually store entire list all at once. Good to use in for loops when there is a big range and little memory.

5.5 Instructions LOGO

`degrees()` fixe l'unité d'angle en degrés
`radians()` fixe l'unité d'angle en radians
`reset()` efface l'écran et réinitialise les variables
`clear()` efface l'écran
`up()` lève le crayon
`down()` abaisse le crayon
`forward(d)` avance d'une distance *d*
`backward(d)` recule d'une distance *d*
`left(a)` tourne sur la gauche d'un angle *a*
`right(a)` tourne sur la droite d'un angle *a*
`goto(x,y)` déplace le crayon à la position (*x*,*y*)
`towards(x,y)` oriente vers le point de coordonnées (*x*,*y*)
`setheading(a)` oriente d'un angle *a* par rapport à l'axe des *x*
`position()` donne la position (*x*,*y*) du crayon
`heading()` donne l'orientation *a* du déplacement
`circle(r)` trace un cercle de rayon *r*
`circle(r,a)` trace un arc de cercle de rayon *r* et d'angle au sommet *a*.

Pour utiliser ces fonctions, il faut les importer depuis le module `turtle` :

```
>>> from turtle import *
```

5.6 Les séquences en PYTHON

Les principales opérations sur les séquences en PYTHON (`list`, `tuple`, `str`) sont listées dans les tableaux ci-dessous.

Operation on sequences (list, tuple, str)	Result
<code>x in s</code> <code>x not in s</code>	True if an item of <code>s</code> is equal to <code>x</code> , else False False if an item of <code>s</code> is equal to <code>x</code> , else True
<code>s1 + s2</code> <code>s * n, n*s</code>	the concatenation of <code>s1</code> and <code>s2</code> <code>n</code> copies of <code>s</code> concatenated
<code>s[i]</code> <code>s[i:j[:step]]</code>	<code>i</code> 'th item of <code>s</code> , origin 0 Slice of <code>s</code> from <code>i</code> (included) to <code>j</code> (excluded). Optional <code>step</code> value, possibly negative (default : 1)
<code>len(s)</code> <code>min(s)</code> <code>max(s)</code>	Length of <code>s</code> Smallest item of <code>s</code> Largest item of <code>s</code>

Operation on list	Result
<code>s[i] = x</code> <code>s[i:j [:step]] = t</code> <code>del s[i :j [:step]]</code>	item <code>i</code> of <code>s</code> is replaced by <code>x</code> slice of <code>s</code> from <code>i</code> to <code>j</code> is replaced by <code>t</code> same as <code>s[i :j] = []</code>
<code>s.count(x)</code> <code>s.index(x[,start[,stop]])</code>	returns number of <code>i</code> 's for which <code>s[i] == x</code> returns smallest <code>i</code> such that <code>s[i] == x</code> . <code>start</code> and <code>stop</code> limit search to only part of the list
<code>s.append(x)</code> <code>s.extend(x)</code> <code>s.insert(i, x)</code> <code>s.remove(x)</code> <code>s.pop([i])</code>	same as <code>s[len(s) : len(s)] = [x]</code> same as <code>s[len(s) :len(s)]= x</code> same as <code>s[i :i] = [x]</code> if <code>i >= 0</code> . <code>i == -1</code> inserts before the last element same as <code>del s[s.index(x)]</code> same as <code>x = s[i] ; del s[i] ; return x</code>
<code>s.reverse()</code> <code>s.sort([cmp])</code>	reverses the items of <code>s</code> in place sorts the items of <code>s</code> in place

Operation on str	Result
<code>s.capitalize()</code> <code>s.center(width[, fillChar=' '])</code>	Returns a copy of <code>s</code> with its first character capitalized, and the rest of the characters lowercased Returns a copy of <code>s</code> centered in a string of length <code>width</code> , surrounded by the appropriate number of <code>fillChar</code> characters
<code>s.count(sub[,start[, end]])</code>	Returns the number of occurrences of substring <code>sub</code> in string <code>s</code>
<code>s.decode([encoding[, errors]])</code>	Returns a unicode string representing the decoded version of str <code>s</code> , using the given codec (<code>encoding</code>). Useful when reading from a file or a I/O function that handles only str. Inverse of <code>encode</code>
<code>s.encode([encoding[, errors]])</code>	Returns a str representing an encoded version of <code>s</code> . Mostly used to encode a unicode string to a str in order to print it or write it to a file (since these I/O functions only accept str). Inverse of <code>decode</code>
<code>s.endswith(suffix[, start[,end]])</code>	Returns True if <code>s</code> ends with the specified <code>suffix</code> , otherwise return False.

<code>s.expandtabs([tabsize])</code>	Returns a copy of <code>s</code> where all tab characters are expanded using spaces
<code>s.find(sub[,start[,end]])</code>	Returns the lowest index in <code>s</code> where substring <code>sub</code> is found. Returns -1 if <code>sub</code> is not found
<code>s.index(sub[,start[,end]])</code>	like <code>find()</code> , but raises <code>ValueError</code> when the substring <code>sub</code> is not found
<code>s.isalnum()</code>	Returns <code>True</code> if all characters in <code>s</code> are alphanumeric, <code>False</code> otherwise
<code>s.isalpha()</code>	Returns <code>True</code> if all characters in <code>s</code> are alphabetic, <code>False</code> otherwise
<code>s.isdigit()</code>	Returns <code>True</code> if all characters in <code>s</code> are digit characters, <code>False</code> otherwise
<code>s.isspace()</code>	Returns <code>True</code> if all characters in <code>s</code> are whitespace characters, <code>False</code> otherwise
<code>s.istitle()</code>	Returns <code>True</code> if string <code>s</code> is a titlecased string, <code>False</code> otherwise
<code>s.islower()</code>	Returns <code>True</code> if all characters in <code>s</code> are lowercase, <code>False</code> otherwise
<code>s.isupper()</code>	Returns <code>True</code> if all characters in <code>s</code> are uppercase, <code>False</code> otherwise
<code>separator.join(seq)</code>	Returns a concatenation of the strings in the sequence <code>seq</code> , separated by string <code>separator</code> , e.g. : " <code>",".join(['A','B','C'])</code> " -> " <code>A,B,C</code> "
<code>s.ljust/rjust/center(width[,fillChar=' '])</code>	Returns <code>s</code> left/right justified/centered in a string of length <code>width</code>
<code>s.lower()</code>	Returns a copy of <code>s</code> converted to lowercase
<code>s.lstrip([chars])</code>	Returns a copy of <code>s</code> with leading <code>chars</code> (default : blank chars) removed
<code>s.partition(separ)</code>	Searches for the separator <code>separ</code> in <code>s</code> , and returns a tuple (<code>head</code> , <code>sep</code> , <code>tail</code>) containing the part before it, the separator itself, and the part after it. If the separator is not found, returns <code>s</code> and two empty strings
<code>s.replace(old,new[,maxCount=-1])</code>	Returns a copy of <code>s</code> with the first <code>maxCount</code> (-1 : unlimited) occurrences of substring <code>old</code> replaced by <code>new</code>
<code>s.rfind(sub[,start[,end]])</code>	Returns the highest index in <code>s</code> where substring <code>sub</code> is found. Returns -1 if <code>sub</code> is not found
<code>s.rindex(sub[,start[,end]])</code>	like <code>rfind()</code> , but raises <code>ValueError</code> when the substring <code>sub</code> is not found
<code>s.rpartition(separ)</code>	Searches for the separator <code>separ</code> in <code>s</code> , starting at the end of <code>s</code> , and returns a tuple (<code>tail</code> , <code>sep</code> , <code>head</code>) containing the part before it, the separator itself, and the part after it. If the separator is not found, returns two empty strings and <code>s</code>
<code>s.rstrip([chars])</code>	Returns a copy of <code>s</code> with trailing <code>chars</code> (default : blank chars) removed, e.g. <code>aPath.rstrip('/')</code> will remove the trailing <code>'/'</code> from <code>aPath</code> if it exists
<code>s.split([separator[,maxsplit]])</code>	Returns a list of the words in <code>s</code> , using <code>separator</code> as the delimiter string

<code>s.rsplit([separator[, maxsplit]])</code>	Same as <code>split</code> , but splits from the end of the string
<code>s.splitlines([keepends])</code>	Returns a list of the lines in <code>s</code> , breaking at line boundaries
<code>s.startswith(prefix[, start[,end]])</code>	Returns <code>True</code> if <code>s</code> starts with the specified <code>prefix</code> , otherwise returns <code>False</code> . Negative numbers may be used for <code>start</code> and <code>end</code>
<code>s.strip([chars])</code>	Returns a copy of <code>s</code> with leading and trailing <code>chars</code> (default : blank chars) removed
<code>s.swapcase()</code>	Returns a copy of <code>s</code> with uppercase characters converted to lowercase and vice versa
<code>s.title()</code>	Returns a titlecased copy of <code>s</code> , i.e. words start with uppercase characters, all remaining cased characters are lowercase
<code>s.translate(table[, deletechars])</code>	Returns a copy of <code>s</code> mapped through translation table <code>table</code>
<code>s.upper()</code>	Returns a copy of <code>s</code> converted to uppercase
<code>s.zfill(width)</code>	Returns the numeric string left filled with zeros in a string of length <code>width</code>

5.7 Les fichiers en PYTHON

Les principales opérations sur les fichiers en PYTHON (type `file`) sont listées dans le tableau ci-dessous.

`open(filename[,mode='r',[bufsize]])` returns a new file object. `filename` is the file name to be opened. `mode` indicates how the file is to be opened ('r', 'w', 'a', '+', 'b', 'U'). `bufsize` is 0 for unbuffered, 1 for line buffered, negative or omitted for system default, >1 for a buffer of (about) the given size.

Operation on file	Result
<code>f.close()</code>	Close file <code>f</code>
<code>f.fileno()</code>	Get fileno (fd) for file <code>f</code>
<code>f.flush()</code>	Flush file <code>f</code> 's internal buffer
<code>f.isatty()</code>	<code>True</code> if file <code>f</code> is connected to a tty-like dev, else <code>False</code>
<code>f.next()</code>	Returns the next input line of file <code>f</code> , or raises <code>StopIteration</code> when EOF is hit
<code>f.read([size])</code>	Read at most <code>size</code> bytes from file <code>f</code> and return as a string object. If <code>size</code> omitted, read to EOF
<code>f.readline()</code>	Read one entire line from file <code>f</code> . The returned line has a trailing <code>n</code> , except possibly at EOF. Return '' on EOF
<code>f.readlines()</code>	Read until EOF with <code>readline()</code> and return a list of lines read
<code>for line in f : ...</code>	Iterate over the lines of a file <code>f</code> (using <code>readline</code>)
<code>f.seek(offset[, whence=0])</code>	Set file <code>f</code> 's position <code>whence == 0</code> then use absolute indexing <code>whence == 1</code> then <code>offset</code> relative to current pos <code>whence == 2</code> then <code>offset</code> relative to file end
<code>f.tell()</code>	Return file <code>f</code> 's current position (byte offset)

<code>f.truncate([size])</code>	Truncate <code>f</code> 's <code>size</code> . If <code>size</code> is present, <code>f</code> is truncated to (at most) that <code>size</code> , otherwise <code>f</code> is truncated at current position (which remains unchanged)
<code>f.write(str)</code> <code>f.writelines(list)</code>	Write string <code>str</code> to file <code>f</code> Write list of strings to file <code>f</code> . No EOL are added

5.8 Utilitaire pydoc

Cette annexe est un extrait du site officiel PYTHON concernant `pydoc` : <http://docs.python.org/lib/module-pydoc.html>. La figure 5.2 ci-dessous illustre son utilisation.

Fig. 5.2 : DOCUMENTATION EN PYTHON

```
$ pydoc fibo
Help on module fibo :

NAME
    fibo
FILE
    /home/info/S1/cours/fonctions/fibo.py

FUNCTIONS
    fibonacci(n)
        u = fibonacci(n)
        est le nombre de Fibonacci
        à l'ordre n si n :int >= 0
    >>> fibonacci(0)
    1
    >>> fibonacci(2)
    2
    >>> fibonacci(9)
    55
```

pydoc – Documentation generator and online help system

The `pydoc` module automatically generates documentation from Python modules. The documentation can be presented as pages of text on the console, served to a Web browser, or saved to HTML files.

The built-in function `help()` invokes the online help system in the interactive interpreter, which uses `pydoc` to generate its documentation as text on the console. The same text documentation can also be viewed from outside the PYTHON interpreter by running `pydoc` as a script at the operating system's command prompt. For example, running

```
pydoc sys
```

at a shell prompt will display documentation on the `sys` module, in a style similar to the manual pages shown by the Unix `man` command. The argument to `pydoc` can be the name of a function, module, or package, or a dotted reference to a class, method, or function within a module or module in a package. If the argument to `pydoc` looks like a path (that is, it contains the path separator for your operating system, such as a slash in Unix), and refers to an existing PYTHON source file, then documentation is produced for that file.

Specifying a `-w` flag before the argument will cause HTML documentation to be written out to a file in the current directory, instead of displaying text on the console.

Specifying a `-k` flag before the argument will search the synopsis lines of all available modules for the keyword given as the argument, again in a manner similar to the Unix `man` command. The synopsis line of a module is the first line of its documentation string.

You can also use `pydoc` to start an HTTP server on the local machine that will serve documentation to visiting Web browsers. `pydoc -p 1234` will start a HTTP server on port 1234, allowing you to browse the documentation at `http://localhost:1234/` in your preferred Web browser. `pydoc -g` will start the server and additionally bring up a small Tkinter-based graphical interface to help you search for documentation pages.

When `pydoc` generates documentation, it uses the current environment and path to locate modules. Thus, invoking `pydoc spam` documents precisely the version of the module you would get if you started the PYTHON interpreter and typed "import spam".

5.9 Transformation d'une récursivité terminale

Quel que soit le problème à résoudre, on a le choix entre l'écriture d'une fonction itérative et celle d'une fonction récursive. Si le problème admet une décomposition récurrente naturelle, le programme récursif est alors une simple adaptation de la décomposition choisie. C'est le cas des fonctions `factorielle` et `fibonacci` par exemple. L'approche récursive présente cependant des inconvénients : certains langages n'admettent pas la récursivité (comme le langage machine !) et elle est souvent coûteuse en mémoire comme en temps d'exécution. On peut pallier ces inconvénients en transformant la fonction récursive en fonction itérative : c'est toujours possible.

Considérons une procédure `f` à récursivité terminale écrite en pseudo-code :

<pre>def f(x): if cond: arret else: instructions f(g(x)) return</pre>	<p><code>x</code> représente ici la liste des arguments de la fonction, <code>cond</code> une condition portant sur <code>x</code>, <code>instructions</code> un bloc d'instructions qui constituent le traitement de base de la fonction <code>f</code>, <code>g(x)</code> une transformation des arguments et <code>arret</code> l'instruction de terminaison (clause d'arrêt) de la récurrence.</p>
---	--

Elle est équivalente à la procédure itérative suivante :

```
def f(x):
    while not cond:
        instructions
        x = g(x)
    arret
    return
```

Illustrons cette transformation à l'aide de la fonction qui calcule le pgcd de 2 entiers.

```
def pgcd(a,b):
    if b == 0: return a
    else:
        pass # ne fait rien
        return pgcd(b,a%b)
```

```
>>> pgcd(12,18)
6
```

<code>x</code>	→ <code>a,b</code>
<code>cond</code>	→ <code>b == 0</code>
<code>arret</code>	→ <code>return a</code>
<code>instructions</code>	→ <code>pass</code>
<code>x = g(x)</code>	→ <code>a,b = b,a%b</code>

```
def pgcd(a,b):
    while not (b == 0):
        pass
        a,b = b,a%b
    return a
```

```
>>> pgcd(12,18)
6
```

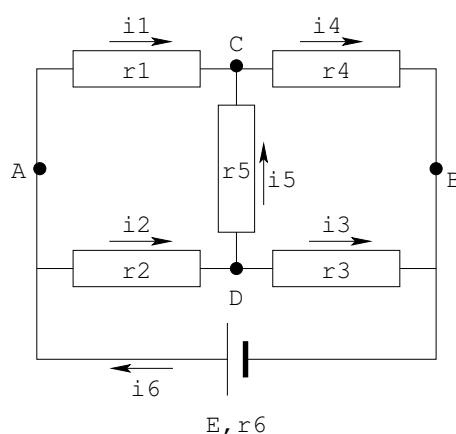
La méthode précédente ne s'applique qu'à la récursivité terminale.

5.10 Méthode d'élimination de GAUSS

L'objectif est ici de résoudre dans \mathbb{R} un système de n équations linéaires à n inconnues, homogènes ou non homogènes, du type $A \cdot x = b$:

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + \cdots + a_{0(n-1)}x_{(n-1)} = b_0 \\ a_{10}x_0 + a_{11}x_1 + \cdots + a_{1(n-1)}x_{(n-1)} = b_1 \\ \cdots + \cdots + \cdots + \cdots = \cdots \\ a_{(n-1)0}x_0 + a_{(n-1)1}x_1 + \cdots + a_{(n-1)(n-1)}x_{(n-1)} = b_{(n-1)} \end{cases} \quad (5.1)$$

Fig. 5.3 : PONT DE WHEATSTONE



$r_1 = 10\Omega$	$i_4 = i_1 + i_5$
$r_2 = 10\Omega$	$i_6 = i_1 + i_2$
$r_3 = 5\Omega$	$i_2 = i_3 + i_5$
$r_4 = 20\Omega$	$10i_1 = 10i_2 + 10i_5$
$r_5 = 10\Omega$	$10i_5 = 5i_3 - 20i_4$
$r_6 = 10\Omega$	$12 - 10i_6 = 10i_2 + 5i_3$
$E = 12V$	

De nombreux exemples traités dans les enseignements scientifiques conduisent à la nécessité de résoudre un système de n équations linéaires à n inconnues, homogènes ou non homogènes, du type $A \cdot x = b$. La figure 5.3 propose à titre d'exemple le cas du pont de Wheatstone en électricité. Les ponts ont été utilisés pour la mesure des résistances, inductances et capacités jusqu'à ce que les progrès en électronique les rendent obsolètes en métrologie. Toutefois la structure en pont reste encore utilisée dans de nombreux montages.

La première méthode généralement utilisée pour trouver la solution d'un système d'équations linéaires tel que le système (5.1) est celle qui consiste à éliminer les inconnues (x_i) en combinant les équations. Pour illustrer cette méthode, nous commencerons par étudier un exemple simple pouvant s'effectuer *à la main*, puis nous passerons au cas général pour présenter la méthode d'élimination de GAUSS.

Etude d'un cas particulier

Considérons le système suivant :

$$\begin{cases} x_0 + x_1 + x_2 = 1 \\ 2x_0 + 4x_1 + 8x_2 = 10 \\ 3x_0 + 9x_1 + 27x_2 = 33 \end{cases}$$

Pour résoudre un tel système, l'idée de base est de triangulariser le système de telle manière qu'il soit possible de remonter les solutions par substitutions successives.

La première étape consiste à éliminer le terme en x_0 des 2^{ème} et 3^{ème} équations. Cherchons tout d'abord à éliminer le terme en x_0 de la 2^{ème} équation. Pour cela nous multiplions la 1^{ère} équation par le coefficient de x_0 de la 2^{ème} équation ($a_{10} = 2$). On obtient le nouveau système :

$$\begin{cases} 2x_0 + 2x_1 + 2x_2 = 2 \\ 2x_0 + 4x_1 + 8x_2 = 10 \\ 3x_0 + 9x_1 + 27x_2 = 33 \end{cases}$$

On soustrait alors la première équation de la deuxième équation, ce qui conduit au système :

$$\begin{cases} 2x_0 + 2x_1 + 2x_2 = 2 \\ \quad 2x_1 + 6x_2 = 8 \\ 3x_0 + 9x_1 + 27x_2 = 33 \end{cases}$$

On recommence l'opération pour éliminer le terme en x_0 de la 3^{ème} équation. Pour cela, on ramène à 1 le coefficient de x_0 dans la première équation en divisant cette équation par 2 ($a_{00} = 2$), puis nous multiplions la 1^{ère} équation par le coefficient de x_0 de la 3^{ème} équation ($a_{30} = 3$).

$$\begin{cases} 3x_0 + 3x_1 + 3x_2 = 3 \\ \quad 2x_1 + 6x_2 = 8 \\ 3x_0 + 9x_1 + 27x_2 = 33 \end{cases}$$

On soustrait ensuite la 1^{ère} équation de la 3^{ème} équation :

$$\begin{cases} 3x_0 + 3x_1 + 3x_2 = 3 \\ \quad 2x_1 + 6x_2 = 8 \\ \quad 6x_1 + 24x_2 = 30 \end{cases}$$

On obtient un nouveau système linéaire dans lequel seule la première équation contient un terme en x_0 . L'équation utilisée (ici la 1^{ère} équation) pour éliminer une inconnue dans les équations qui suivent (ici les 2^{ème} et 3^{ème} équations) est appelée l'équation-pivot. Dans l'équation-pivot choisie, le coefficient de l'inconnue qui est éliminée dans les autres équations est appelé le pivot de l'équation (ici a_{00}).

La deuxième étape consiste à éliminer le terme en x_1 de la troisième équation en utilisant la deuxième équation comme équation-pivot. On ramène à 1 le coefficient de x_1 dans la 2^{ème} équation en divisant l'équation par 2 ($a_{11} = 2$), puis on la multiplie par 6 ($a_{31} = 6$) pour que les 2^{ème} et 3^{ème} équations aient le même terme en x_1 . Tout revient à multiplier la 2^{ème} équation par 3 ($a_{31}/a_{11} = 6/2 = 3$) :

$$\begin{cases} 3x_0 + 3x_1 + 3x_2 = 3 \\ \quad 6x_1 + 18x_2 = 24 \\ \quad 6x_1 + 24x_2 = 30 \end{cases}$$

Il reste à soustraire la 2^{ème} équation de la 3^{ème} pour éliminer le terme en x_1 de la 3^{ème} équation :

$$\begin{cases} 3x_0 + 3x_1 + 3x_2 = 3 \\ 6x_1 + 18x_2 = 24 \\ 6x_2 = 6 \end{cases}$$

On obtient ainsi un système triangulaire d'équations linéaires dont on peut calculer directement la valeur de x_2 par la 3^{ème} équation : $6x_2 = 6 \Rightarrow x_2 = 1$. On porte cette valeur de x_2 dans la deuxième équation, ce qui nous permet de calculer x_1 : $6x_1 + 18 \cdot 1 = 24 \Rightarrow x_1 = 1$. En reportant les valeurs de x_2 et x_1 dans la première équation, on en déduit la valeur de x_0 : $3x_0 + 3 \cdot 1 + 3 \cdot 1 = 3 \Rightarrow x_0 = -1$. On vérifie simplement que les valeurs obtenues sont solutions du système initial :

$$\begin{cases} x_0 + x_1 + x_2 = 1 \\ 2x_0 + 4x_1 + 8x_2 = 10 \\ 3x_0 + 9x_1 + 27x_2 = 33 \end{cases} \quad \left(\begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{pmatrix} \right) \cdot \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 10 \\ 33 \end{pmatrix}$$

Etude du cas général

De manière générale, la méthode précédente consiste à réduire le système de n équations à n inconnues à un système triangulaire équivalent qui peut être ensuite résolu facilement par substitutions. En quelque sorte le système (5.1) doit être transformé en un système équivalent du type :

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0(n-1)}x_{(n-1)} = b_0 \\ a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1(n-1)}^{(1)}x_{(n-1)} = b_1^{(1)} \\ \quad + a_{22}^{(2)}x_2 + \cdots + a_{2(n-1)}^{(2)}x_{(n-1)} = b_2^{(2)} \\ \quad \quad \quad \cdots + \quad \quad \quad \cdots = \cdots \\ a_{(n-1)(n-1)}^{(n-2)}x_{(n-1)} = b_{(n-1)}^{(n-2)} \end{cases} \quad (5.2)$$

où l'indice supérieur désigne le nombre d'étapes à la suite desquelles est obtenu le coefficient considéré.

L'équation de rang 0 du système (5.1) est d'abord divisée par le coefficient a_{00} de x_0 (supposé non nul). On obtient :

$$x_0 + \frac{a_{01}}{a_{00}}x_1 + \frac{a_{02}}{a_{00}}x_2 + \cdots + \frac{a_{0(n-1)}}{a_{00}}x_{(n-1)} = \frac{b_0}{a_{00}} \quad (5.3)$$

Cette équation (5.3) est alors multipliée par a_{10} , coefficient de x_0 dans la deuxième équation du système (5.1). Le résultat est ensuite soustrait de la deuxième équation du système (5.1), ce qui élimine x_0 dans la deuxième équation. D'une manière générale on multiplie la relation (5.3) par a_{i0} , coefficient de x_0 dans l'équation de rang i du système (5.1) et on retranche le résultat obtenu de cette même $i^{\text{ème}}$ équation. A la fin x_0 est éliminé de toutes les équations, excepté de la première, et on obtient le système ainsi transformé :

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0(n-1)}x_{(n-1)} = b_0 \\ a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1(n-1)}^{(1)}x_{(n-1)} = b_1^{(1)} \\ a_{21}^{(1)}x_1 + a_{22}^{(1)}x_2 + \cdots + a_{2(n-1)}^{(1)}x_{(n-1)} = b_2^{(1)} \\ a_{31}^{(1)}x_1 + a_{32}^{(1)}x_2 + \cdots + a_{3(n-1)}^{(1)}x_{(n-1)} = b_3^{(1)} \\ \quad \quad \quad \cdots + \quad \quad \quad \cdots + \quad \quad \quad \cdots = \cdots \\ a_{(n-1)1}^{(1)}x_1 + a_{(n-1)2}^{(1)}x_2 + \cdots + a_{(n-1)(n-1)}^{(1)}x_{(n-1)} = b_{(n-1)}^{(1)} \end{cases} \quad (5.4)$$

L'équation de rang 1 dans le nouveau système (5.4) devient alors l'équation-pivot et $a_{11}^{(1)}$ le pivot de l'équation. De la même façon on élimine x_1 des équations du rang 2 au rang $n-1$ dans le système (5.4), et on obtient :

$$\left\{ \begin{array}{l} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0(n-1)}x_{(n-1)} = b_0 \\ \quad a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1(n-1)}^{(1)}x_{(n-1)} = b_1^{(1)} \\ \quad \quad a_{22}^{(2)}x_2 + \cdots + a_{2(n-1)}^{(2)}x_{(n-1)} = b_2^{(2)} \\ \quad \quad \quad a_{32}^{(2)}x_2 + \cdots + a_{3(n-1)}^{(2)}x_{(n-1)} = b_3^{(2)} \\ \quad \quad \quad \quad \cdots + \cdots + \cdots = \cdots \\ \quad \quad \quad \quad \quad a_{(n-1)2}^{(2)}x_2 + \cdots + a_{(n-1)(n-1)}^{(2)}x_{(n-1)} = b_{(n-1)}^{(2)} \end{array} \right. \quad (5.5)$$

L'équation de rang 2 dans le nouveau système (5.5) fait office à son tour d'équation pivot, et ainsi de suite jusqu'à obtenir le système triangulaire (5.2). Une fois obtenu ce système triangulaire, on calcule directement la valeur de $x_{(n-1)}$ par la dernière relation du système triangulaire. En portant cette valeur dans la relation précédente, on calculera $x_{(n-2)}$, et ainsi de suite en remontant le système triangulaire (5.2). A chaque étape k , on a donc les relations suivantes :

1. Triangularisation

$$\left\{ \begin{array}{l} a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ip}^{(k-1)}}{a_{pp}^{(k-1)}} \cdot a_{pj}^{(k-1)} \\ b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ip}^{(k-1)}}{a_{pp}^{(k-1)}} \cdot b_p^{(k-1)} \end{array} \right. \quad \text{avec} \quad \left\{ \begin{array}{l} k : \text{étape} \\ p : \text{rang du pivot} \\ p < i \leq (n-1) \\ p \leq j \leq (n-1) \end{array} \right.$$

2. Remontée par substitutions

$$\left\{ \begin{array}{l} x_{(n-1)} = \frac{b_{(n-1)}}{a_{(n-1)(n-1)}} \\ x_i = \frac{b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j}{a_{ii}} \end{array} \right. \quad \text{avec } 0 \leq i < (n-1)$$

Jusqu'à présent, on a admis que le pivot au cours du processus de triangularisation était non nul ($a_{pp} \neq 0$). Si ce n'est pas le cas, on doit permuter l'équation-pivot avec une autre ligne dans laquelle le pivot est différent de 0. Il se peut également que le pivot, sans être nul, soit très petit et l'on a intérêt là-aussi à interchanger les lignes comme pour le cas d'un pivot nul. En fait, pour augmenter la précision de la solution obtenue, on a toujours intérêt à utiliser la ligne qui a le plus grand pivot.

Cette méthode est connue sous le nom de méthode d'élimination de GAUSS (également connue sous le nom de méthode de triangularisation de GAUSS ou méthode du pivot de GAUSS). Elle fut nommée ainsi en l'honneur du mathématicien allemand JOHANN CARL FRIEDRICH GAUSS (1777–1855), mais elle est connue des Chinois depuis au moins le 1^{er} siècle de notre ère. Elle est référencée dans le livre chinois « Jiuzhang suanshu » où elle est attribuée à CHANG TS'ANG chancelier de l'empereur de Chine au 2^{ème} siècle avant notre ère.

Jeu de tests

L'algorithme de résolution de tels systèmes linéaires pourra être testé à l'aide des systèmes suivants :

Test	Système $A \cdot x = b$		Solution exacte
	A	b	
1	$\begin{pmatrix} 4 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1/4 \end{pmatrix}$
2	$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$
3	$\begin{pmatrix} 2 & -1 & 2 \\ 1 & 10 & -3 \\ -1 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 5 \\ -3 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}$

Test	Système $A \cdot x = b$		Solution exacte
	A	b	
4	$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$	$\begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$
5	$\begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix}$	$\begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$	—
6	$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$	$\begin{pmatrix} 32.01 \\ 22.99 \\ 33.01 \\ 30.99 \end{pmatrix}$	—

Les tests 5 et 6 sont des variations du test 4 : $A_5 = A_4 + \Delta A$ et $b_6 = b_4 + \Delta b$. Ces 2 tests sont effectués pour évaluer les conséquences sur la solution x d'une perturbation ΔA de A_4 et d'une perturbation Δb de b_4 .

$$\Delta A = \begin{pmatrix} 0 & 0 & 0.1 & 0.2 \\ 0.08 & 0.04 & 0 & 0 \\ 0 & -0.02 & -0.11 & 0 \\ -0.01 & -0.01 & 0 & -0.02 \end{pmatrix}$$

$$\Delta b = \begin{pmatrix} 0.01 \\ -0.01 \\ 0.01 \\ -0.01 \end{pmatrix}$$

Test	Système $A \cdot x = b$							
	A							b
7	$\begin{pmatrix} 1 & 0 & 0 & - & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 10 & -10 & 0 & 0 & -10 & 0 \\ 0 & 0 & 5 & -20 & -10 & 0 \\ 0 & 10 & 5 & 0 & 0 & 10 \end{pmatrix}$							$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 12 \end{pmatrix}$
8	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2\pi & 4\pi^2 & 8\pi^3 & 16\pi^4 & 32\pi^5 & 64\pi^6 & 128\pi^7 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4\pi & 12\pi^2 & 32\pi^3 & 80\pi^4 & 192\pi^5 & 448\pi^6 \\ 1 & \pi/2 & \pi^2/4 & \pi^3/8 & \pi^4/16 & \pi^5/32 & \pi^6/64 & \pi^7/128 \\ 0 & \pi/2 & \pi & 3\pi^2/4 & \pi^3/2 & 5\pi^4/16 & 3\pi^5/16 & 7\pi^6/64 \\ 1 & \pi & \pi^2 & \pi^3 & \pi^4 & \pi^5 & \pi^6 & \pi^7 \\ 0 & \pi & 2\pi & 3\pi^2 & 4\pi^3 & 5\pi^4 & 6\pi^5 & 7\pi^6 \end{pmatrix}$							$\begin{pmatrix} 4 \\ 4 \\ 0 \\ 0 \\ 2 \\ -2 \\ 1 \\ 0 \end{pmatrix}$

Le test 7 correspond à l'exemple du pont de Wheatstone (figure 5.3). Le test 8 correspond à la détermination de la forme d'une came rotative qui doit mettre en mouvement un axe suivant une loi horaire donnée.

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

2.11	Catégorie sportive	16
2.12	Dessin d'étoiles (1)	16
2.13	Fonction factorielle	16
2.14	Fonction sinus	17
2.15	Algorithme d'Euclide	17
2.16	Division entière	17
2.17	Affichage inverse	17
2.18	Parcours inverse	17
2.19	Suite arithmétique (2)	17
2.20	Dessin d'étoiles (2)	17
2.21	Opérateurs booléens dérivés (2)	17
2.22	Damier	17
2.23	Trace de la fonction factorielle	18
2.24	Figure géométrique	18
2.25	Suite arithmétique (3)	18
2.26	QCM (2)	18
2.27	Unité de longueur	20
2.28	Permutation circulaire (2)	21
2.29	Séquence d'affectations (2)	21
2.30	Circuits logiques (2)	21
2.31	Alternative simple et test simple	22
2.32	Racines du trinôme	22
2.33	Séquences de tests	22
2.34	Racine carrée entière	23
2.35	Exécutions d'instructions itératives	23
2.36	Figures géométriques	24
2.37	Suites numériques	25
2.38	Calcul vectoriel	25
2.39	Prix d'une photocopie	25
2.40	Calcul des impôts	25
2.41	Développements limités	25
2.42	Tables de vérité	25
2.43	Dessins géométriques	26
2.44	Police d'assurance	26
2.45	Zéro d'une fonction	26
3.1	Codage des entiers positifs (1)	29
3.2	Codage d'un nombre fractionnaire	29
3.3	Décodage base $b \rightarrow$ décimal	30
3.4	Codage des entiers positifs (2)	30
3.5	Une spécification, des implémentations	30
3.6	Passage par valeur	30
3.7	Valeurs par défaut	30
3.8	Portée des variables	30
3.9	Tours de Hanoï à la main	31
3.10	Pgcd et ppcm de 2 entiers (1)	31
3.11	Somme arithmétique	31
3.12	Courbes fractales	31
3.13	QCM (3)	32

3.14	Passage des paramètres	33
3.15	Portée des variables (2)	33
3.16	Suite géométrique	34
3.17	Puissance entière	34
3.18	Coefficients du binôme	34
3.19	Fonction d'Ackerman	34
3.20	Addition binaire	34
3.21	Complément à 2	35
3.22	Codage-décodage des réels	35
3.23	Intégration numérique	36
3.24	Tracés de courbes paramétrées	37
4.1	Distance de 2 points de l'espace	39
4.2	Opérations sur les n-uplets	39
4.3	Pgcd et ppcm de 2 entiers (2)	39
4.4	Opérations sur les chaînes	39
4.5	Inverser une chaîne	39
4.6	Caractères, mots, lignes d'une chaîne	39
4.7	Opérations sur les listes (1)	39
4.8	Opérations sur les listes (2)	39
4.9	Sélection d'éléments	40
4.10	Opérations sur les piles	40
4.11	Opérations sur les files	40
4.12	Produit de matrices	40
4.13	Annuaire téléphonique	40
4.14	Recherche dichotomique	40
4.15	Liste ordonnée	40
4.16	Tri d'un annuaire téléphonique	41
4.17	Complexité du tri par sélection	41
4.18	Tri par insertion	41
4.19	Comparaison d'algorithmes (1)	41
4.20	QCM (4)	41
4.21	Génération de séquences	43
4.22	Application d'une fonction à tous les éléments d'une liste	43
4.23	Que fait cette procédure ?	43
4.24	Codes ASCII et chaînes de caractères	44
4.25	Opérations sur les matrices	44
4.26	Recherche d'un motif	44
4.27	Recherche de toutes les occurrences	44
4.28	Tri bulles	44
4.29	Méthode d'élimination de GAUSS	44
4.30	Comparaison d'algorithmes de recherche.	45
4.31	Comparaison d'algorithmes de tri	45