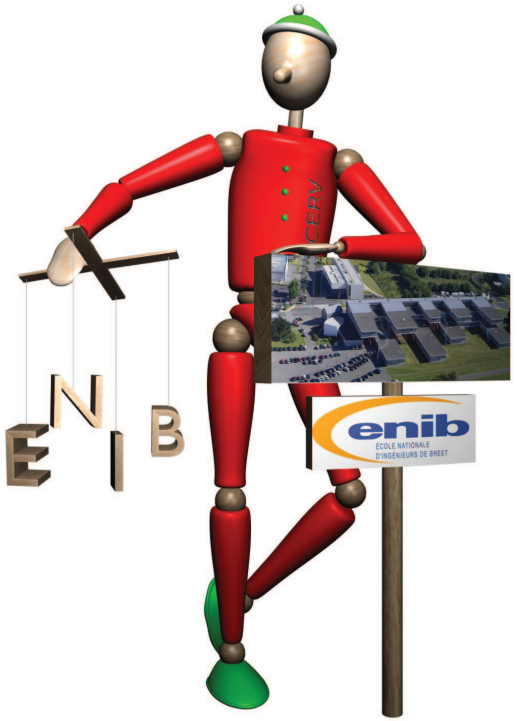
— Cours d'Informatique S1 —

Initiation à l'algorithmique

Jacques TISSEAU

LISYC EA 3883 UBO-ENIB-ENSIETA
Centre Européen de Réalité Virtuelle
tisseau@enib.fr



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

Avec la participation de Romain Bénard, Stéphane Bonneaud, Cédric Buche, Gireg Desmeulles, Eric Maisel, Aléxis Nédélec, Marc Parenthoën et Cyril Septseault.

version du 01 septembre 2009

Sommaire

« Chaque programme d'ordinateur est un modèle, forgé par l'esprit, d'un processus réel ou imaginaire. Ces processus, qui naissent de l'expérience et de la pensée de l'homme, sont innombrables et complexes dans leurs détails. A tout moment, ils ne peuvent être compris que partiellement. Ils ne sont que rarement modélisés d'une façon satisfaisante dans nos programmes informatiques. Bien que nos programmes soient des ensembles de symboles ciselés avec soin, des mosaïques de fonctions entrecroisées, ils ne cessent d'évoluer. Nous les modifions au fur et à mesure que notre perception du modèle s'approfondit, s'étend et se généralise, jusqu'à atteindre un équilibre métastable aux frontières d'une autre modélisation possible du problème. L'ivresse joyeuse qui accompagne la programmation des ordinateurs provient des allers-retours continuels, entre l'esprit humain et l'ordinateur, des mécanismes exprimés par des programmes et de l'explosion de visions nouvelles qu'ils apportent. Si l'art traduit nos rêves, l'ordinateur les réalise sous la forme de programmes ! »

Abelson H., Sussman G.J. et Sussman J., [1]

Chapitre 1

Introduction générale

Sommaire

Fig. 1.1 : Définitions de l'Académie (1)
INFORMATIQUE *n. f. et adj. XXe siècle. Dérivé d'information sur le modèle de mathématique, électronique. 1. N. f. Science du traitement rationnel et automatique de l'information ; l'ensemble des applications de cette science. 2. Adj. Qui se rapporte à l'informatique. Système informatique, ensemble des moyens qui permettent de conserver, de traiter et de transmettre l'information.*
INSTRUCTION *n. f. XIVe siècle. Emprunté du latin instructio, « action d'adapter, de ranger », puis « instruction ». Ordre, indication qu'on donne à quelqu'un pour la conduite d'une affaire ; directive, consigne. Le plus souvent au pluriel. Des instructions verbales, écrites. Donner des instructions à ses collaborateurs. Instructions pour la mise en marche d'un appareil. Par anal. INFORM. Consigne formulée dans un langage de programmation, selon un code.*
LOGICIEL *n. m. XXe siècle. Dérivé de logique. INFORM. Ensemble structuré de programmes remplissant une fonction déterminée, permettant l'accomplissement d'une tâche donnée.*
MATÉRIEL *adj. et n. XIIIe siècle. Emprunté du latin materialis, de même sens. INFORM. Ensemble des éléments physiques employés pour le traitement des données, par opposition aux logiciels.*
ORDINATEUR *n. m. XVe siècle, au sens de « celui qui institue quelque chose » ; XXe siècle, au sens actuel. Emprunté du latin ordinator, « celui qui règle, met en ordre ; ordonnateur ». Équipement informatique comprenant les organes nécessaires à son fonctionnement autonome, qui assure, en exécutant les instructions d'un ensemble structuré de programmes, le traitement rapide de données codées sous forme numérique qui peuvent être conservées et transmises.*

1.1 Contexte scientifique

1.1.1 Informatique

Le terme INFORMATIQUE est un néologisme proposé en 1962 par Philippe Dreyfus pour caractériser le traitement automatique de l'information : il est construit sur la contraction de l'expression « information automatique ». Ce terme a été accepté par l'Académie française en avril 1966, et l'informatique devint alors officiellement la science du traitement automatique de l'information, où l'information est considérée comme le support des connaissances humaines et des communications dans les domaines techniques, économiques et sociaux (figure 1.1). Le mot INFORMATIQUE n'a pas vraiment d'équivalent aux Etats-Unis où l'on parle de *Computing Science* (science du calcul) alors que *Informatics* est admis par les Britanniques.

Définition 1.1 : INFORMATIQUE
L'informatique est la science du traitement automatique de l'information.

L'informatique traite de deux aspects complémentaires : les programmes immatériels (logiciel, *software*) qui décrivent un traitement à réaliser et les machines (matériel, *hardware*) qui exécutent ce traitement. Le matériel est donc l'ensemble des éléments physiques (microprocesseur, mémoire, écran, clavier, disques durs...) utilisés pour traiter les données. Dans ce contexte, l'ordinateur est un terme générique qui désigne un équipement informatique permettant de traiter des informations selon des séquences d'instructions (les programmes) qui constituent le logiciel. Le terme ORDINATEUR a été proposé par le philologue Jacques Perret en avril 1955 en réponse à une demande d'IBM France qui estimait le mot « calculateur » (*computer*) bien trop restrictif en regard des possibilités de ces machines (voir la proposition de Jacques Perret en annexe 1.6.1 page 31).

Définition 1.2 : MATÉRIEL
Le matériel informatique est un ensemble de dispositifs physiques utilisés pour traiter automatiquement des informations.

Définition 1.3 : LOGICIEL
Le logiciel est un ensemble structuré d'instructions décrivant un traitement d'informations à faire réaliser par un matériel informatique.

Un ordinateur n'est rien d'autre qu'une machine effectuant des opérations simples sur des séquences de signaux électriques, lesquels sont conditionnés de manière à ne pouvoir prendre

que deux états seulement (par exemple un potentiel électrique maximum ou minimum). Ces séquences de signaux obéissent à une logique binaire du type « tout ou rien » et peuvent donc être considérés conventionnellement comme des suites de nombres ne prenant jamais que les deux valeurs 0 et 1 : un ordinateur est donc incapable de traiter autre chose que des nombres binaires. Toute information d'un autre type doit être convertie, ou codée, en format binaire. C'est vrai non seulement pour les données que l'on souhaite traiter (les nombres, les textes, les images, les sons, les vidéos, etc.), mais aussi pour les programmes, c'est-à-dire les séquences d'instructions que l'on va fournir à la machine pour lui dire ce qu'elle doit faire avec ces données.

L'architecture matérielle d'un ordinateur repose sur le modèle de Von Neumann (figure 1.2) qui distingue classiquement 4 parties (figure 1.3) :

1. L'unité arithmétique et logique, ou unité de traitement, effectue les opérations de base.
2. L'unité de contrôle séquence les opérations.
3. La mémoire contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire vive volatile (programmes et données en cours de fonctionnement) et mémoire de masse permanente (programmes et données de base de la machine).
4. Les entrées-sorties sont des dispositifs permettant de communiquer avec le monde extérieur (écran, clavier, souris...).

Les 2 premières parties sont souvent rassemblées au sein d'une même structure : le microprocesseur (la « puce »), unité centrale de l'ordinateur.

Dans ce cours, nous ne nous intéresserons qu'aux aspects logiciels de l'informatique.

1.1.2 Algorithmique

Exemple 1.1 : Mode d'emploi d'un télécopieur

Extrait du mode d'emploi d'un télécopieur concernant l'envoi d'un document.

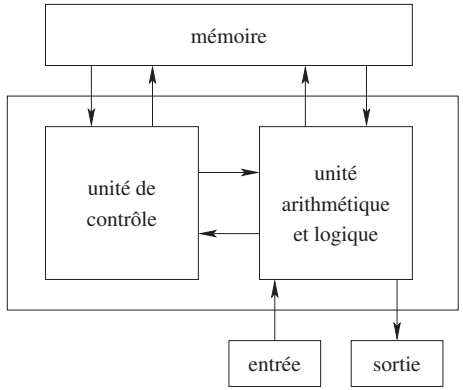
1. *Insérez le document dans le chargeur automatique.*
2. *Composez le numéro de fax du destinataire à l'aide du pavé numérique.*
3. *Enfoncez la touche* ENVOI *pour lancer l'émission.*

Ce mode d'emploi précise comment envoyer un fax. Il est composé d'une suite ordonnée d'instructions (insérez..., composez..., enfoncez...) qui manipulent des données (document, chargeur

Fig. 1.2 : John Von Neumann (1903-1957)

Mathématicien américain d'origine hongroise : il a donné son nom à l'architecture de von Neumann utilisée dans la quasi totalité des ordinateurs modernes (voir figure 1.3 ci-dessous).

Fig. 1.3 : Architecture de Von Neumann



automatique, numéro de fax, pavé numérique, touche ENVOI) pour réaliser la tâche désirée (envoi d'un document).

Chacun a déjà été confronté à de tels documents pour faire fonctionner un appareil plus ou moins réticent et donc, consciemment ou non, a déjà exécuté un algorithme (ie. exécuter la suite d'instructions dans l'ordre annoncé, figure 1.4). ■TD1.1

Définition 1.4 : ALGORITHME
Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.

Exemple 1.2 : TROUVER SON CHEMIN
Extrait d'un dialogue entre un touriste égaré et un autochtone.
- *Pourriez-vous m'indiquer le chemin de la gare, s'il vous plait ?*
- *Oui bien sûr : vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous.*
- *Merci.*

Dans ce dialogue, la réponse de l'autochtone est la description d'une suite ordonnée d'instructions (allez tout droit, prenez à gauche, prenez la troisième à droite) qui manipulent des données (carrefour, rues) pour réaliser la tâche désirée (aller à la gare). Ici encore, chacun a déjà été confronté à ce genre de situation et donc, consciemment ou non, a déjà construit un algorithme dans sa tête (ie. définir la suite d'instructions pour réaliser une tâche). Mais quand on définit un algorithme, celui-ci ne doit contenir que des instructions compréhensibles par celui qui devra l'exécuter (des humains dans les 2 exemples précédents). ■TD1.2

Dans ce cours, nous devrons apprendre à définir des algorithmes pour qu'ils soient compréhensibles — et donc exécutables — par un ordinateur.

Définition 1.5 : ALGORITHMIQUE
L'algorithmique est la science des algorithmes.

L'algorithmique s'intéresse à l'art de construire des algorithmes ainsi qu'à caractériser leur validité, leur robustesse, leur réutilisabilité, leur complexité ou leur efficacité.

Définition 1.6 : VALIDITÉ D'UN ALGORITHME
La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.

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.

Fig. 1.4 : DÉFINITIONS DE L'ACADÉMIE (2)
ALGORITHME *n. m. XIIIe siècle, augorisme. Altération, sous l'influence du grec arithmos, « nombre », d'algorisme, qui, par l'espagnol, remonte à l'arabe Al-Khuwarizmi, surnom d'un mathématicien. Méthode de calcul qui indique la démarche à suivre pour résoudre une série de problèmes équivalents en appliquant dans un ordre précis une suite finie de règles.*
DONNÉE *n. f. XIIIe siècle, au sens de « distribution, aumône » ; XVIIIe siècle, comme terme de mathématiques. Participe passé féminin substantivé de donner au sens de « indiquer, dire ». 1. Fait ou principe indiscuté, ou considéré comme tel, sur lequel se fonde un raisonnement ; constatation servant de base à un examen, une recherche, une découverte.* INFORM. *Représentation d'une information sous une forme conventionnelle adaptée à son exploitation.*

Si l'on reprend l'exemple 1.2 de l'algorithme de recherche du chemin de la gare, l'étude de sa validité consiste à s'assurer qu'on arrive effectivement à la gare en exécutant scrupuleusement les instructions dans l'ordre annoncé.

Définition 1.7 : ROBUSTESSE D'UN ALGORITHME
La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.

Dans l'exemple 1.2, la question de la robustesse de l'algorithme se pose par exemple si le chemin proposé a été pensé pour un piéton, alors que le « touriste égaré » est en voiture et que la « troisième à droite » est en sens interdit.

Définition 1.8 : RÉUTILISABILITÉ D'UN ALGORITHME
La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.

L'algorithme de recherche du chemin de la gare est-il réutilisable tel quel pour se rendre à la mairie ? A priori non, sauf si la mairie est juste à côté de la gare.

Définition 1.9 : COMPLEXITÉ D'UN ALGORITHME
La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.

Si le « touriste égaré » est un piéton, la complexité de l'algorithme de recherche de chemin peut se compter en nombre de pas pour arriver à la gare.

Définition 1.10 : EFFICACITÉ D'UN ALGORITHME
L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute.

N'existerait-il pas un raccourci piétonnier pour arriver plus vite à la gare ? ■TD1.3

L'algorithmique permet ainsi de passer d'un problème à résoudre à un algorithme qui décrit la démarche de résolution du problème. La programmation a alors pour rôle de traduire cet algorithme dans un langage « compréhensible » par l'ordinateur afin qu'il puisse exécuter l'algorithme automatiquement (figure 1.5).

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.

Fig. 1.5 : DU PROBLÈME AU CODE SOURCE

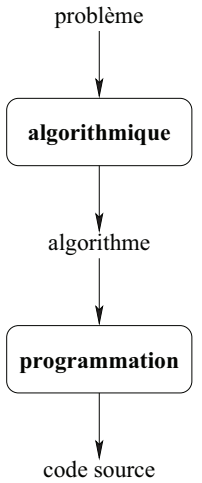


Fig. 1.6 : Définitions de l'Académie (3)
LANGAGE. *n. m. XIIe siècle. Dérivé de langue. Système de signes, de symboles, élaboré à partir des langues naturelles et constituant un code qui les remplace dans certains cas déterminés (on parle aussi de langage symbolique). Le langage mathématique. Langage logique, fondé sur la logique formelle. Langage informatique. Langage de programmation.*
BIT *n. m. XXe siècle. Mot anglo-américain, contraction de binary digit, « chiffre binaire ». Chacun des deux chiffres, 0 et 1, de la numération binaire. En informatique, le bit est l'unité élémentaire d'information appelée aussi élément binaire.*
OCTET *n. m. XXe siècle. Dérivé savant du latin octo, « huit ». Unité d'information composée de huit bits.*

Fig. 1.7 : Clé USB (Universal Serial Bus)

Dispositif matériel contenant une mémoire de masse (une mémoire flash ou un mini disque dur).

1.1.3 Programmation

Un algorithme exprime la structure logique d'un programme informatique et de ce fait est indépendant du langage de programmation utilisé. Par contre, la traduction de l'algorithme dans un langage particulier dépend du langage choisi et sa mise en œuvre dépend également de la plateforme d'exécution.

La programmation d'un ordinateur consiste à lui « expliquer » en détail ce qu'il doit faire, en sachant qu'il ne « comprend » pas le langage humain, mais qu'il peut seulement effectuer un traitement automatique sur des séquences de 0 et de 1. Un programme n'est rien d'autre qu'une suite d'instructions, encodées en respectant de manière très stricte un ensemble de conventions fixées à l'avance par un langage informatique (figure 1.6). La machine décode alors ces instructions en associant à chaque « mot » du langage informatique une action précise. Le programme que nous écrivons dans le langage informatique à l'aide d'un éditeur (une sorte de traitement de texte spécialisé) est appelé programme source (ou code source).

Le seul « langage » que l'ordinateur puisse véritablement « comprendre » est donc très éloigné de ce que nous utilisons nous-mêmes. C'est une longue suite de 0 et de 1 (les « bits », *binary digit*) traités par groupes de 8 (les « octets », *byte*), 16, 32, ou même 64 (figure 1.6).

Définition 1.11 : bit
Un bit est un chiffre binaire (0 ou 1). C'est l'unité élémentaire d'information.

Définition 1.12 : octet
Un octet est une unité d'information composée de 8 bits.

Exemple 1.3 : Description de clé USB
Sur le descriptif commercial d'une clé USB (figure 1.7), on peut lire :
- *Type de produit : Lecteur flash USB*
- *Taille du module : 512 Mo*
- *Type d'interface : Hi-Speed USB*

La « taille du module » est la capacité de stockage de la clé qui vaut ici 512 Mo (mégaoctets). Le préfixe « méga » est un multiplicateur décimal qui vaut 10^6 mais s'il est bien adapté au calcul décimal, il l'est moins au calcul binaire car il ne correspond pas à une puissance entière de 2. En effet, la puissance x de 2 telle que $2^x = 10^6$ vaut $x = 6 \cdot \dfrac{\log(10)}{\log(2)} \approx 19.93$. On choisit alors la

puissance entière de 2 immédiatement supérieure ($2^{20} = 1\,048\,576$) pour définir le Mo : 1 Mo = 1 048 576 octets. ■ TD1.4

Pour « parler » à un ordinateur, il nous faudra donc utiliser des systèmes de traduction automatiques, capables de convertir en nombres binaires des suites de caractères formant des mots-clés (anglais en général) qui seront plus significatifs pour nous. Le système de traduction proprement dit s'appellera interpréteur ou bien compilateur, suivant la méthode utilisée pour effectuer la traduction (figure 1.8).

Définition 1.13 : COMPILATEUR
Un compilateur est un programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible.

Définition 1.14 : INTERPRÉTEUR
Un interpréteur est un outil informatique (logiciel ou matériel) ayant pour tâche d'analyser et d'exécuter un programme écrit dans un langage source.

On appellera langage de programmation un ensemble de mots-clés (choisis arbitrairement) associé à un ensemble de règles très précises indiquant comment on peut assembler ces mots pour former des « phrases » que l'interpréteur ou le compilateur puisse traduire en langage machine (binaire). Un langage de programmation se distingue du langage mathématique par sa visée opérationnelle (ie. il doit être exécutable par une machine), de sorte qu'un langage de programmation est toujours un compromis entre sa puissance d'expression et sa possibilité d'exécution.

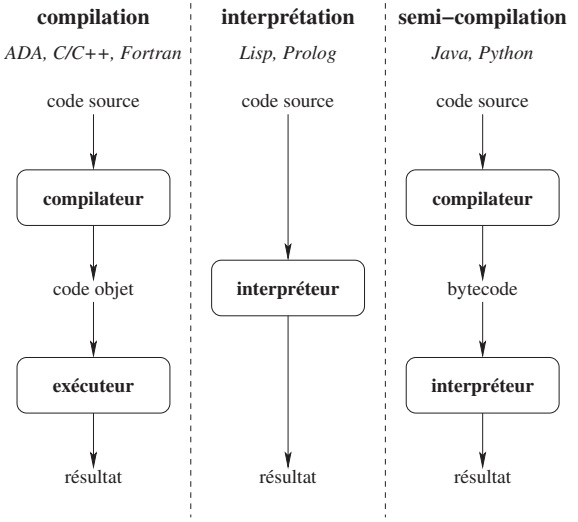
Définition 1.15 : LANGAGE DE PROGRAMMATION
Un langage de programmation est un langage informatique, permettant à un humain d'écrire un code source qui sera analysé par un ordinateur.

Le code source subit ensuite une transformation ou une évaluation dans une forme exploitable par la machine, ce qui permet d'obtenir un programme. Les langages permettent souvent de faire abstraction des mécanismes bas niveaux de la machine, de sorte que le code source représentant une solution puisse être rédigé et compris par un humain.

Définition 1.16 : PROGRAMMATION
La programmation est l'activité de rédaction du code source d'un programme.

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) ?

Fig. 1.8 : DU CODE SOURCE À SON EXÉCUTION



Compilation : La compilation consiste à traduire la totalité du code source en une fois. Le compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes appelé programme objet (ou code objet). Celui-ci peut désormais être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier (« fichier exécutable »). Les langages Ada, C, C++ et Fortran sont des exemples de langages compilés.

Interprétation : L'interprétation consiste à traduire chaque ligne du programme source en quelques instructions du langage machine, qui sont ensuite directement exécutées au fur et à mesure (« au fil de l'eau »). Aucun programme objet n'est généré. L'interpréteur doit être utilisé chaque fois que l'on veut faire fonctionner le programme. Les langages Lisp et Prolog sont des exemples de langages interprétés.

L'interprétation est idéale lorsque l'on est en phase d'apprentissage du langage, ou en cours d'expérimentation sur un projet. Avec cette technique, on peut en effet tester immédiatement toute modification apportée au programme source, sans passer par une phase de compilation qui demande toujours du temps. Mais lorsqu'un projet comporte des fonctionnalités complexes qui doivent s'exécuter rapidement, la compilation est préférable : un programme compilé fonctionnera toujours plus vite que son homologue interprété, puisque dans cette technique l'ordinateur n'a plus à (re)traduire chaque instruction en code binaire avant qu'elle puisse être exécutée.

Semi-compilation : Certains langages tentent de combiner les deux techniques afin de retirer le meilleur de chacune. C'est le cas des langages Python et Java par exemple. De tels langages commencent par compiler le code source pour produire un code intermédiaire, similaire à un langage machine (mais pour une machine virtuelle), que l'on appelle bytecode, lequel sera ensuite transmis à un interpréteur pour l'exécution finale. Du point de vue de l'ordinateur, le bytecode est très facile à interpréter en langage machine. Cette interprétation sera donc beaucoup plus rapide que celle d'un code source.

Le fait de disposer en permanence d'un interpréteur permet de tester immédiatement n'importe quel petit morceau de programme. On pourra donc vérifier le bon fonctionnement de chaque composant d'une application au fur et à mesure de sa construction. L'interprétation du bytecode compilé n'est pas aussi rapide que celle d'un véritable code machine, mais elle est satisfaisante pour de très nombreux programmes.

Dans ce cours, nous choisirons d'exprimer directement les algorithmes dans un langage informatique opérationnel : le langage Python [16], sans passer par un langage algorithmique intermédiaire.

Python : www.python.org

1.2 Objectifs du cours

1.2.1 Objectifs thématiques

L'objectif principal des enseignements d'informatique S1 de l'ENIB est l'acquisition des notions fondamentales de l'algorithmique. Plus précisément, nous étudierons successivement :

1. les instructions de base permettant de décrire les algorithmes : affectation, tests, boucles ;
2. les procédures et les fonctions qui permettent de structurer et de réutiliser les algorithmes ; on parlera alors d'encapsulation, de préconditions, de portée des variables, de passage de paramètres, d'appels de fonctions, de récursivité et de jeux de tests ;
3. les structures de données linéaires : tableaux, listes, piles, files, qui améliorent la structuration des données manipulées par les algorithmes. A cette occasion, on évaluera la complexité et l'efficacité de certains algorithmes utilisant ces structures linéaires.

Ces différentes notions seront mise en œuvre à travers l'utilisation du langage PYTHON. ■TD1.5

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 calculette.*
3. *Quitter* PYTHON.

Ne pas oublier de se déconnecter du poste de travail.

1.2.2 Objectifs pédagogiques

Au cours du semestre S1, nous nous positionnerons principalement sur les 3 premiers niveaux de la taxonomie de Bloom qui constitue une référence pédagogique pour la classification des niveaux d'apprentissage (figure 1.9) : connaissance, compréhension, application. Les 3 derniers niveaux seront plutôt abordés au cours du semestre S2 (analyse, synthèse, évaluation).

1. Connaissance : mémorisation et restitution d'informations dans les mêmes termes.
2. Compréhension : restitution du sens des informations dans d'autres termes.
3. Application : utilisation de règles, principes ou algorithmes pour résoudre un problème, les règles n'étant pas fournies dans l'énoncé.
4. Analyse : identification des parties constituantes d'un tout pour en distinguer les idées.
5. Synthèse : réunion ou combinaison des parties pour former un tout.
6. Evaluation : formulation de jugements qualitatifs ou quantitatifs.

Afin de mieux nous situer par rapport aux différents types de pédagogie associés, nous « filerons » une métaphore musicale inspirée de [3].

Fig. 1.9 : TAXONOMIE DE BLOOM
Benjamin Bloom (1913-1999), psychologue américain spécialisé en pédagogie.

Connaître	*: définir, distinguer, acquérir, identifier, rappeler, reconnaître...*
Comprendre	*: traduire, illustrer, représenter, dire avec ses mots, distinguer, réécrire, réarranger, expliquer, démontrer...*
Appliquer	*: appliquer, généraliser, relier, choisir, développer, utiliser, employer, transférer, classer, restructurer...*
Analyser	*: distinguer, détecter, classer, reconnaître, catégoriser, déduire, discerner, comparer...*
Synthétiser	*: écrire, relater, produire, constituer, transmettre, modifier, créer, proposer, planifier, projeter, spécifier, combiner, classer, formuler...*
Evaluer	*: juger, argumenter, valider, décider, comparer...*

Pédagogie par objectifs : Le solfège est l'étude des principes élémentaires de la musique et de sa notation : le musicien « fait ses gammes » et chaque exercice a un objectif précis pour évaluer l'apprentissage du « langage musical ». Il en va de même pour l'informaticien débutant confronté à l'apprentissage d'un langage algorithmique.

Pédagogie par l'exemple : L'apprentissage des grands classiques permet au musicien de s'approprier les bases du solfège en les appliquant à ces partitions connues et en les (re)jouant lui-même. L'informaticien débutant, en (re)codant lui-même des algorithmes bien connus, se constituera ainsi une base de réflexes de programmation en « imitant » ces algorithmes.

Pédagogie de l'erreur : Les bogues (*bugs*) sont à l'informaticien ce que les fausses notes sont aux musiciens : des erreurs. Ces erreurs sont nécessaires dans l'acquisition de la connaissance. Un élève a progressé si, après s'être trompé, il peut reconnaître qu'il s'est trompé, dire où et pourquoi il s'est trompé, et comment il recommencerait sans produire les mêmes erreurs.

Pédagogie par problèmes : Connaissant « ses » classiques et les bases du solfège, le musicien devenu plus autonome peut envisager sereinement la création de ses propres morceaux. Le développement d'un projet informatique ambitieux sera « mis en musique » au semestre S2.

Dans ce cours, nous adopterons ces différentes stratégies pédagogiques sans oublier qu'en informatique on apprend toujours mieux en faisant par soi-même.

Remarque 1.1 : *L'annexe A page 207 présente quelques grands classiques « historiques ». On trouvera également une liste d'algorithmes classiques sur le site du National Institute of Standards and Technology* (NIST : www.nist.gov/dads).

1.2.3 Objectifs comportementaux

Nous cherchons à développer trois « qualités » comportementales chez l'informaticien débutant : la rigueur, la persévérance et l'autonomie.

Rigueur : Un ordinateur est une machine qui exécute vite et bien les instructions qu'on lui a « apprises ». Mais elle ne sait pas interpréter autre chose : même mineure, une erreur provoque le dysfonctionnement de la machine.

Exemple 1.4 : Erreur de syntaxe en langage C
Pour un « ; » oublié dans un programme C*, le code source n'est pas compilable et le compilateur nous avertit par ce type de message :*

```
fichier.c : In function ‘main’ :
fichier.c :7 : error : syntax error before "printf"
```

Même si un humain peut transiger sur le « ; », la machine ne le peut pas : l'ordinateur ne se contente pas de l'« à peu près ». La respect des consignes, la précision et l'exactitude sont donc de rigueur en informatique ! ■TD1.6

Persévérance : Face à l'intransigeance de la machine, le débutant est confronté à ses nombreuses erreurs (les siennes, pas celles de la machine !) et sa tendance naturelle est de passer à autre chose. Mais le papillonnage (ou *zapping*) est une très mauvaise stratégie en informatique : pour s'exécuter correctement, un programme doit être finalisé. L'informatique nécessite d'« aller au bout des choses ». ■TD1.7

Autonomie : Programmer soi-même les algorithmes qu'on a définis est sans doute le meilleur moyen pour mieux assimiler les principales structures algorithmiques et pour mieux comprendre ses erreurs en se confrontant à l'intransigeante impartialité de l'ordinateur, véritable « juge de paix » des informaticiens. Par ailleurs, l'évolution continuelle et soutenue des langages de programmation et des ordinateurs nécessitent de se tenir à jour régulièrement et seule l'autoformation systématique permet de « ne pas perdre pied ». ■TD1.8

Pratique personnelle et autoformation constituent ainsi deux piliers de l'autonomisation nécessaire de l'apprenti informaticien.

1.3 Organisation du cours

1.3.1 Présentiel

Les enseignements d'informatique S1 de l'ENIB sont dispensés lors de 42h de séances de cours-TD et de séances de laboratoire.

- Les cours-TD ont lieu 1 semaine sur 2 à raison de 3h par semaine, soit 21h de cours-TD sur toute la durée du semestre. Ils se déroulent dans une salle banalisée.
- Les séances de laboratoire ont lieu 1 semaine sur 2 en alternance avec les cours-TD à raison de 3h par semaine, soit 21h de laboratoire dans le semestre. Elles se déroulent en salle informatique.

1.3.2 Documents

Les principaux documents accompagnant les cours sont de 2 types : les supports de cours et les notes de cours.

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
>>>
```

De quelle erreur de syntaxe s'agit-il ?

TD 1.7 : Dessins sur la plage : persévérance
Finir l'algorithme suivant qui cherche à dessiner un losange sur la plage.

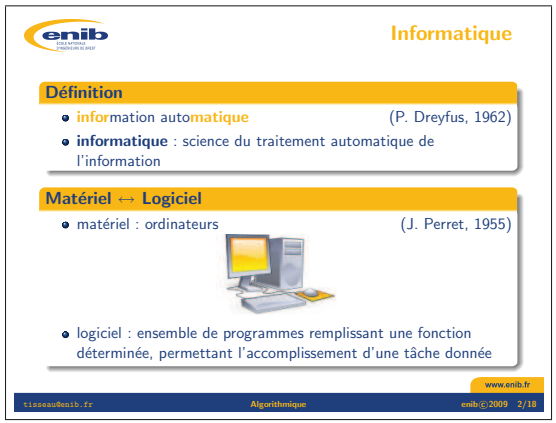
1. *avance de 10 pas,*
2. *tourne à gauche d'un angle de* 60°*,*

⋮

TD 1.8 : Autonomie
Trouver les définitions du mot autonomie et de son contraire (de son antonyme).

Remarque 1.2 : *Un semestre s'étale sur 14 semaines d'enseignements, 7 semaines consacrées au cours et 7 semaines aux TD.*

Fig. 1.10 : TRANSPARENT DE COURS



TD 1.9 : SITE WEB D'INFORMATIQUE S1
Se connecter sur le site WEB *du cours d'informatique S1 de l'ENIB et vérifier que ces notes de cours sont bien disponibles sur le site au format* `pdf`.

TD 1.10 : EXEMPLE DE CONTRÔLE D'ATTENTION (1)
Répondre de mémoire aux questions suivantes (ie. sans rechercher les solutions dans les pages précédentes).

1. *Quels sont les 3 principaux thèmes informatiques abordés ?*
2. *Quelles sont les 4 principales stratégies pédagogiques suivies ?*
3. *Quelles sont les 3 principales qualités comportementales recherchées ?*

Support de cours : il s'agit de la copie papier des transparents projetés en présentiel (figure 1.10).

Notes de cours : il s'agit de notes qui complètent et précisent certains points présentés en cours. Ces notes proposent également les exercices de travaux dirigés qui sont étudiés en cours et au laboratoire. Le document que vous êtes en train de lire constitue le premier chapitre des notes du cours d'informatique S1 de l'ENIB. Il y en aura 3 autres sur les sujets suivants :

1. les instructions de base (chapitre 2 page 39),
2. les procédures et les fonctions (chapitre 3 page 99),
3. les structures de données linéaires (chapitre 4 page 157).

Un site WEB permet de retrouver ces documents au format `pdf` (*Portable Document Format*) ainsi que d'autres documents tels que le planning prévisionnel des cours et des contrôles, des exemples corrigés d'évaluation, les notes aux différents contrôles ou encore des liens vers des sites pertinents pour le cours. Un forum de discussions professeurs↔élèves est également ouvert sur ce site. ■TD1.9

La consultation régulière du site est indispensable pour se tenir au courant des dernières évolutions du cours : en cas d'ambiguïté, ce sont les informations de ce site qui feront foi.

1.3.3 Evaluations des apprentissages

L'évaluation des connaissances et des compétences acquises par les étudiants repose sur 4 types de contrôle : les contrôles d'attention, les contrôles de TD, les contrôles d'autoformation et les contrôles de compétences.

Contrôle d'attention : il s'agit d'un QCM (questionnaire à choix multiples) auquel il faut répondre individuellement sans document, en 5' en fin de cours, et dont les questions portent sur des points abordés pendant ce cours. Ce type de contrôle teste directement l'acquisition de connaissances au sens du « connaître » de la classification de Bloom (section 1.2.2 page 11). Il permet d'évaluer « à chaud » la capacité d'attention des étudiants et les incite à une présence attentive afin de bénéficier au maximum des heures de présence aux cours. ■TD1.10

Des exemples de QCM sont systématiquement proposés dans les notes de cours comme par exemple celui du TD 1.19 page 20 pour cette introduction générale.

Contrôle de TD : il s'agit ici d'inciter les étudiants à préparer activement les séances de laboratoire. En début de chaque séance de laboratoire, chaque binôme doit répondre sans document en 10' aux questions d'un exercice de TD. L'exercice est choisi aléatoirement parmi les exercices de TD situés en marge des notes de cours et se rapportant au thème du TD. ■TD1.11

Il concerne le « comprendre » et l'« appliquer » de la taxonomie de Bloom (section 1.2.2 page 11).

Contrôle d'autoformation : un contrôle d'autoformation porte sur un thème prévu à l'avance et que l'étudiant doit approfondir par lui-même. Les thèmes retenus pour l'autoformation en S1 sont par exemple le calcul booléen, le codage des nombres ou encore la recherche d'éléments dans un tableau de données. ■TD1.12

Ces contrôles se déroulent pendant les séances de cours : ce sont des écrits individuels de 30' sans document qui concernent le « connaître », le « comprendre » et l'« appliquer » de la taxonomie de Bloom.

Contrôle de compétences : les contrôles de compétences (ou DS) durent 80' pendant une séance de cours. Plus longs, ils permettent d'aborder l'un des 3 derniers niveaux de la classification de Bloom (analyser, synthétiser, évaluer) comme le font les exercices de la section 1.5.4 page 23. ■TD1.13

Quel que soit le type de contrôle, un exercice cherche à évaluer un objectif particulier. Aussi, la notation exprimera la distance qui reste à parcourir pour atteindre cet objectif (figure 1.11) :

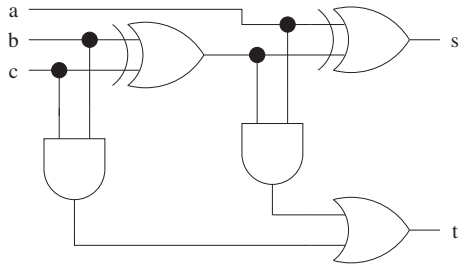
0 : « en plein dans le mille ! » → l'objectif est atteint
1 : « pas mal ! » → on se rapproche de l'objectif
2 : « juste au bord de la cible ! » → on est encore loin de l'objectif
3 : « la cible n'est pas touchée ! » → l'objectif n'est pas atteint

Ainsi, et pour changer de point de vue sur la notation, le contrôle est réussi lorsqu'on a 0 ! Il n'y a pas non plus de 1/2 point ou de 1/4 de point : le seul barème possible ne comporte que 4 niveaux : 0, 1, 2 et 3. On ne cherche donc pas à « grappiller » des points :

- on peut avoir 0 (objectif atteint) et avoir fait une ou deux erreurs bénignes en regard de l'objectif recherché ;
- on peut avoir 3 (objectif non atteint) et avoir quelques éléments de réponse corrects mais sans grand rapport avec l'objectif.

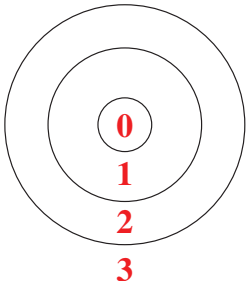
TD 1.11 : Exemple de contrôle de TD
Répondre aux questions du TD 1.10 situé dans la marge de la page 14.

TD 1.12 : Exemple de contrôle d'autoformation (1)
Etablir la table de vérité du circuit logique ci-dessous où a, b et c sont les entrées, s et t les sorties.



TD 1.13 : Exemple de contrôle des compétences
Répondre aux questions du TD 1.28 de la page 25.

Fig. 1.11 : Métaphore de la cible



Remarque 1.3 : *Une absence à un contrôle conduit à la note 4 (« la cible n'est pas visée »).*

1.3.4 Evaluation des enseignements

En fin de semestre, les étudiants organisent de manière anonyme et dans le respect des personnes, une évaluation individuelle des enseignements. Elle est structurée en 2 parties : un questionnaire de 10 à 20 questions (voir un exemple en annexe 1.6.2 page 33) auxquelles il faut répondre selon une grille pré-définie et une partie « expression libre » que l'étudiant remplit, ou non, à son gré.

L'ensemble des fiches d'évaluation est remis à l'équipe pédagogique d'Informatique S1, qui après en avoir pris connaissance, organise une rencontre spécifique avec les étudiants. Cette évaluation a pour objectif l'amélioration de la qualité des enseignements et de la pédagogie à partir de la perception qu'en ont les étudiants.

1.4 Méthodes de travail

Remarque 1.4 : *Ce document comporte 259 pages structurées en 4 chapitres, 3 annexes, 4 index et une bibliographie. Il propose 47 définitions, 86 figures, 39 exemples, 79 remarques, 128 exercices et 5 contrôles types corrigés. En moyenne, au cours des 14 semaines que dure le cours d'informatique S1 de l'ENIB, le travail personnel hebdomadaire consiste donc à lire entre 15 et 20 pages de ce cours en retenant 3 à 4 définitions et en faisant entre 7 et 10 exercices.*

Il ne s'agit pas ici d'imposer des méthodes de travail, mais de fournir des pistes pour ceux qui en cherchent. Dans tous les cas, l'expérience montre que :

1. la seule présence, même attentive et active, aux séances de cours et de laboratoire ne suffit pas : il faut prévoir un temps de travail personnel qui, selon l'étudiant et la matière, peut aller de 50% à 150% du temps de présence en cours ;
2. la régularité dans le travail personnel est un gage d'apprentissage plus efficace.

Le calendrier prévisionnel des enseignements et des contrôles associés est distribué en début de semestre (un exemple de planning est donné en annexe 1.6.3 page 34). ■TD1.14
Les documents de cours sont distribués au moins 15 jours avant les séances correspondantes (sauf en ce qui concerne les 2 premières semaines de cours évidemment). Par ailleurs, à tout moment, le calendrier et les documents sont disponibles sur le site Web d'Informatique S1.

TD 1.14 : Nombre de contrôles
Après consultation du calendrier prévisionnel de l'annexe 1.6.3 page 34, donner le nombre et le type de contrôles prévus au calendrier du semestre.

1.4.1 Avant, pendant et après le cours

Préparation : Certains cours débutent par un contrôle d'autoformation (voir section 1.3.3) dont le thème est en rapport avec certains aspects du cours qui suivra immédiatement. Il est donc nécessaire d'avoir étudié avant et par soi-même le sujet du contrôle.

En général, on trouvera les informations nécessaires soit directement sur le site d'Informatique S1, soit dans les références bibliographiques données en fin des notes de cours (voir

par exemple les références de ce chapitre en page 271), soit dans les polycopiés d'autres cours de l'ENIB (mathématiques, électronique, productique, microprocesseurs...), soit encore sur internet en s'assurant de la qualité du site (préférer des sites universitaires ou d'écoles dont l'activité principale est d'enseigner ces matières).

Par exemple, il est nécessaire d'avoir assimilé les principes du calcul booléen pour maîtriser les tests et les boucles du langage algorithmique. C'est pourquoi, une autoformation est imposée sur ce domaine déjà connu des étudiants. Un contrôle-type d'autoformation sur le calcul booléen pourrait par exemple être composé des TD 1.12 page 15 et 1.15 ci-contre. ■TD1.15

Pour chaque contrôle d'autoformation, un exemple corrigé est disponible sur le site d'Informatique S1. Il est donc fortement recommandé de travailler ce contrôle-type : après avoir revu par soi-même les principales notions du domaine, faire le contrôle sans consulter au préalable la solution proposée.

Participation : Par respect pour les autres participants, la ponctualité est de rigueur pour l'étudiant comme pour le professeur. Mais assister à un cours n'a d'intérêt que dans le cas d'une présence attentive et soutenue : le temps passé en cours pour assimiler les nouvelles notions sera gagné sur le temps de travail personnel futur nécessaire à leur assimilation. Chaque page des supports de cours est constituée d'une copie d'un transparent de cours dans la demi-page supérieure alors que la demi-page inférieure est volontairement laissée vierge pour que l'étudiant prenne des notes pendant le cours (figure 1.12).

La prise de note systématique est en effet un facteur qui favorise l'attention. Un contrôle de type QCM en fin de cours évaluera l'attention des étudiants (voir section 1.3.3). ■TD1.16

Le cours est illustré par des exemples et des exercices : à ces occasions, c'est une participation active de l'étudiant qui est attendue dans une démarche volontaire d'assimilation du cours « à chaud ».

Appropriation : Dans la mesure du possible, il faut relire ses propres notes ainsi que les notes de cours le soir même afin de « fixer » les principales idées du cours. La révision proprement dite peut venir ultérieurement quelques jours plus tard (et non quelques semaines ni quelques mois après).

Les définitions, comme toute définition, sont à apprendre par cœur . Une technique possible est de lire 2 fois de suite à voix haute la définition en détachant distinctement les différentes expressions (exemple : « l'informatique » « est la science » « du traitement » « automatique » « de l'information »), puis l'écrire de mémoire.

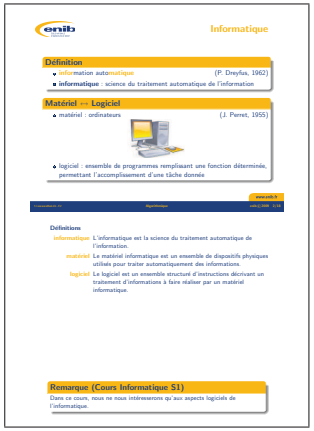
TD 1.15 : Exemple de contrôle d'autoformation (2)
Exprimer en la développant la négation des expressions booléennes suivantes :

1. `(0 < x) and (x < 3)`
2. `(x < -2) or (x > 4)`
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 ?*

Fig. 1.12 : Support de cours



Pour réviser le cours, faire systématiquement les TD au moment où ils sont référencés dans les notes par le symbole ■ . C'est particulièrement vrai avec les exemples pour lesquels sont associés des exercices en marge des notes de cours (exemple : l'exemple 1.1 de la page 5 et le TD 1.1 associé en marge de la même page). Lorsque l'exemple est un algorithme, il faut systématiquement se mettre mentalement à la place de la machine qui va les exécuter (on parle alors d'« empathie numérique ») afin de vérifier le résultat obtenu. Pour cela, il faut être méthodique et rigoureux. Et petit à petit, à force de pratique, l'expérience fera qu'on « verra » le résultat produit par les instructions au fur et à mesure qu'on les écrit. Naturellement, cet apprentissage est long, et demande des heures de travail patient. Aussi, dans un premier temps, il faut éviter de sauter les étapes : la vérification méthodique, pas à pas, de chacun des algorithmes représente plus de la moitié du travail à accomplir [5].

1.4.2 Avant, pendant et après le laboratoire

Préparation : Faire les exercices situés dans les marges de ces notes de cours est non seulement une façon de réviser son cours (voir section 1.4.1) mais aussi de préparer les séances de laboratoire. Pour ces notes de cours, la liste complète des exercices est donnée en annexe B page 223. Un de ces exercices choisi aléatoirement fera l'objet d'une évaluation en début de chaque séance de TD (voir section 1.3.3). ■TD1.17

Tous ces exercices ne nécessitent pas une longue phase de réflexion comme les TD 2.7 ou 1.13 (→ 1.28). Certains au contraire ne présentent aucune difficulté particulière comme les TD 1.14 et 1.17 qui demandent simplement de compter les contrôles et les exercices. D'autres comme les TD 1.10 et 1.16 font appel à la mémoire, d'autres encore à la recherche d'informations comme les TD 1.4 et 1.8, ou à une mise en œuvre pratique comme les TD 1.5 et 1.9.

TD 1.17 : NOMBRES D'EXERCICES DE TD
Combien d'exercices y avait-il à faire avant celui-ci ?

Participation : Ici encore, par respect pour les autres participants, la ponctualité est de rigueur pour l'étudiant comme pour le professeur.

En informatique, lorsqu'on travaille en binôme, il faut régulièrement alterner les rôles entre l'« écrivain » qui manipule clavier et souris et le « lecteur » qui vérifie la production de l'écrivain. A la fin du semestre, chaque étudiant doit être devenu un « lecteur-écrivain ». La pratique est en effet nécessaire pour « apprivoiser » l'environnement de travail afin que la machine devienne « transparente » et que seul subsiste le problème algorithmique à résoudre. ■TD1.18

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 ?*

Pour certains exercices, la solution est donnée dans les notes de cours (en section 1.5.5 page 26 pour ce chapitre). Pendant les séances de laboratoire, il faut donc savoir « jouer le jeu » pour ne pas simplement « recopier » la solution proposée (il existe d'ailleurs d'autres solutions pour la plupart des exercices).

Un apprenti programmeur est toujours confronté à ses propres erreurs (revoir le TD 1.6 page 13 par exemple). En général, ce sont des erreurs simples à corriger à condition de lire les messages d'erreur et de faire l'effort de les comprendre avant d'appeler le professeur « à l'aide ».

Exemple 1.5 : Erreur de nom en Python

Voici une erreur classique d'une variable non correctement initialisée en Python *:*

```
>>> print(x)
Traceback (most recent call last) :
  File "<stdin>", line 1, in ?
NameError : name 'x' is not defined
>>>
```

En Python, la dernière ligne du message d'erreur est la plus « parlante » au débutant.

Appropriation : Avant le cours suivant, il faut refaire les TD qui ont posé des problèmes et faire les exercices qui n'ont pas pu être abordés en séance de laboratoire (les solutions de ces exercices complémentaires sont données dans les notes de cours).

1.4.3 Apprendre en faisant

1. En bas de la page 17, il est dit :

   « Les définitions, comme toute définition, sont à apprendre par cœur. »

   Connaissez-vous les 16 définitions introduites dans ce chapitre ? Elles sont clairement identifiables grâce au mot-clé « **Définition** ».
2. En haut de la page 18, il est dit :

   « Pour réviser le cours, faire systématiquement les TD au moment où ils sont référencés dans les notes par le symbole ■. »

   Avez-vous cherché à résoudre les 18 exercices de TD proposés jusqu'à présent ? Ils sont clairement identifiables grâce au mot-clé « **TD** » situé dans la marge.

Il n'y a pas de miracle, c'est votre travail personnel qui est le meilleur gage de vos apprentissages. On apprend toujours mieux en faisant par soi-même.

Remarque 1.5 : *La dernière phrase de cette section a déjà été écrite dans le texte qui précède. A propos de quoi ? En quelle page ?*

1.5 Exercices complémentaires

1.5.1 Connaître

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*

Remarque 1.6 : *Parmi les 4 items de la question ci-contre, un seul item définit l'informatique, les 3 autres définissent d'autres sciences. Lesquelles ?*

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 héxadécimal*
   (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*

Remarque 1.7 : *Parmi les 4 items de la question ci-contre, un seul item définit la validité d'un algorithme, les 3 autres se rapportent à d'autres propriétés des algorithmes. Lesquelles ?*

TD 1.20 : Puissance de calcul

Donner l'ordre de grandeur en instructions par seconde des machines suivantes (voir [6] par exemple) :

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 (voir [6] par exemple) :

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.*

Remarque 1.8 : *TD 1.21 : voir aussi TD 1.4*

1.5.2 Comprendre

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 ?*

Remarque 1.9 : *TD 1.22 : voir aussi TD 1.1*

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.

Remarque 1.10 : *TD 1.23 : voir aussi TD 1.2*

1.5.3 Appliquer

TD 1.24 : Tracés de polygones réguliers

On cherche à faire dessiner une figure polygonale (figure 1.13) 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).

Remarque 1.11 : *TD 1.24 : voir aussi TD 1.7*

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.*

1.5.4 Analyser

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 : 68×123 $(= 8364)$

multiplicande $M \times 2$	*multiplicateur* $m \div 2$	*reste* $m \bmod 2$	*somme partielle*
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 ».

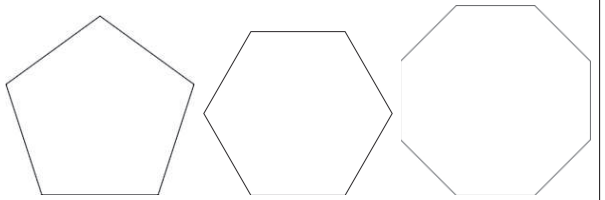
1. 64×96 $(= 6144)$
2. 45×239 $(= 10755)$

TD 1.26 : La multiplication arabe

On considère ici le texte d'Ibn al-Banna concernant la multiplication à l'aide de tableaux [2].

« 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 1.14).

Fig. 1.13 : Pentagone, hexagone, octogone



Remarque 1.12 : *La multiplication en Egypte antique.*

La multiplication « à la russe » est une variante connue d'une technique égyptienne décrite dans le papyrus Rhind (environ -1650). Le scribe Ahmès y expose des problèmes de géométrie et d'arithmétique (qui viennent en partie des Babyloniens) dont cette technique de multiplication.

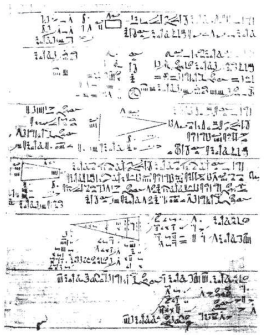


Fig. 1.14 : Tableau d'Ibn al-Banna

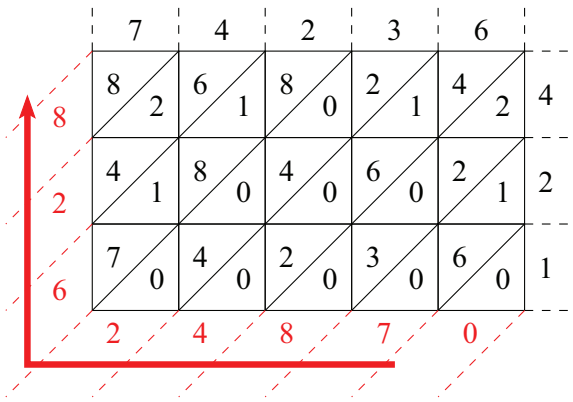
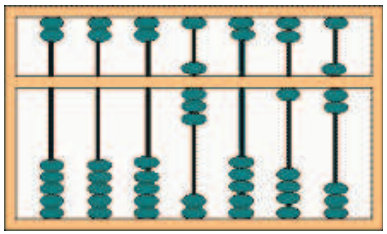


Fig. 1.15 : Boulier chinois

8017 :



Tu places le multiplicande au-dessus du quadrilatère, en faisant correspondre chacune de ses positions à une colonne[1]. 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 ligne[2]. 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.

La somme que tu obtiendras sera le résultat. »

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

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 [2]. Sur chacune de ces tringles, deux étages de boules séparées par une barre transversale peuvent coulisser librement (figure 1.15). 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 1.16) :

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 !*

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

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

[2] L'écriture du nombre s'effectue de bas en haut (exemple : $\begin{smallmatrix}3\\5\\2\end{smallmatrix}$ s'écrira donc $\begin{smallmatrix}2\\5\\3\end{smallmatrix}$).

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 ($1234 = 176 \times 7 + 2$*).*

$$\begin{array}{cccccc} 1 & & 0 & 0 & 0 & 0 \\ \hline 2 & & 1 & 2 & 3 & 4 \end{array} \rightarrow \begin{array}{cccccc} 1 & & 0 & 1 & 1 & 0 \\ \hline 2 & & 1 & 2 & 1 & 2 \end{array}$$

TD 1.28 : Le calcul Shadok

Les cerveaux des Shadoks avaient une capacité tout à fait limitée [15]. 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 1.17). 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

Fig. 1.16 : 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	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 4 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ \hline 2 & 0 & 0 & 0 \end{array}$
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	$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ \hline 2 & 0 & 2 & 0 \end{array}$	$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ \hline 2 & 1 & 0 & 0 \end{array}$

Fig. 1.17 : Les Shadoks : ga bu zo meu

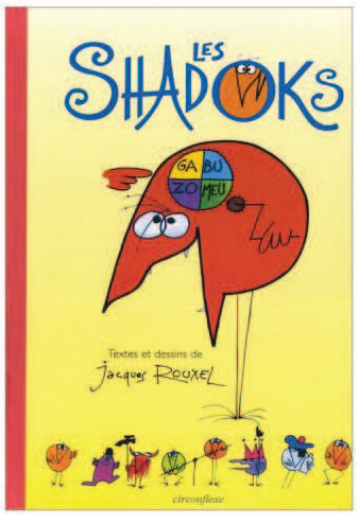


Fig. 1.18 : Les 18 premiers nombres Shadok

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

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 1.18).*

1. *Quels sont les entiers décimaux représentés en « base Shadok » par les expressions suivantes ?*
   - *(a)* ga ga
   - *(b)* bu bu bu
   - *(c)* zo zo zo zo
   - *(d)* meu meu meu meu meu
2. *Effectuer les calculs Shadok suivants.*
   - *(a)* zo zo meu + bu ga meu
   - *(b)* meu ga meu − bu meu ga
   - *(c)* zo meu meu × bu ga meu
   - *(d)* zo zo zo meu ÷ bu ga zo

1.5.5 Solutions des exercices

TD 1.19 : QCM (1).

Les bonnes réponses sont extraites directement du texte de la section 1.1 :
1c, 2d, 3b, 4b, 5d, 6d, 7a, 8c

TD 1.20 : Puissance de calcul.

L'unité de puissance est donné ici en Mips (« million d'instructions par seconde »).

1. le premier micro-ordinateur de type PC : $\approx 10^{-1}$ Mips
2. une console de jeu actuelle : $\approx 10^{4}$ Mips
3. un micro-ordinateur actuel : $\approx 10^{4}$ Mips
4. *Deeper-Blue* : l'ordinateur qui a « battu » Kasparov aux échecs en 1997 : $\approx 10^{7}$ Mips

5. le plus puissant ordinateur actuel : $\approx 10^9$ Mips

TD 1.21 : Stockage de données.

1. une page d'un livre : $\approx$ 50 lignes de 80 caractères = 4 ko
2. une encyclopédie en 20 volumes : $\approx$ 20 volumes de 1000 pages de 50 lignes de 80 caractères = 80 Mo (sans images !)
3. une photo couleur : $\approx$ de 1Mo à 100 Mo selon la qualité
4. une heure de vidéo : $\approx$ de 500 Mo à 2 Go selon la qualité vidéo (DivX, MPEG-2...)
5. une minute de son : $\approx$ de 1 Mo (MP3) à 10 Mo selon la qualité du son
6. une heure de son : $\approx$ de 60 Mo à 600 Mo selon la qualité du son

TD 1.22 : Dessins sur la plage : exécution.

1. On trace un triangle rectangle dont l'hypothénuse fait 5 pas de long (d'après le théorème de Pythagore : $5^2 = 3^2 + 4^2$).
2. On a donc marché $3 + 4 + 5 = 12$ pas.

TD 1.23 : Dessins sur la plage : conception.

Imaginons l'algorithme de tracé suivant :

1. avance de 2 pas,
2. tourne à gauche de 90°,
3. avance de 3 pas,
4. tourne à gauche de 90°,
5. avance de 4 pas,
6. tourne à gauche de 90°,
7. avance de 5 pas,
8. tourne à gauche de 90°,
9. avance de 6 pas.

– validité : on doit au moins vérifier que la figure obtenue à toutes les caractéristiques recherchées : 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 (figure 1.19).

Remarque 1.13 : *Loi de Moore.*
Gordon Earl Moore est le co-fondateur avec Robert Noyce et Andrew Grove de la société Intel en 1968 (fabriquant n°1 mondial de microprocesseurs). En 1965, il expliquait que la complexité des semiconducteurs doublait tous les dix-huit mois à coût constant depuis 1959, date de leur invention. En 1975, il précise sa « première loi » en affirmant que le nombre de transistors des microprocesseurs sur une puce de silicium double tous les deux ans (« deuxième loi »).

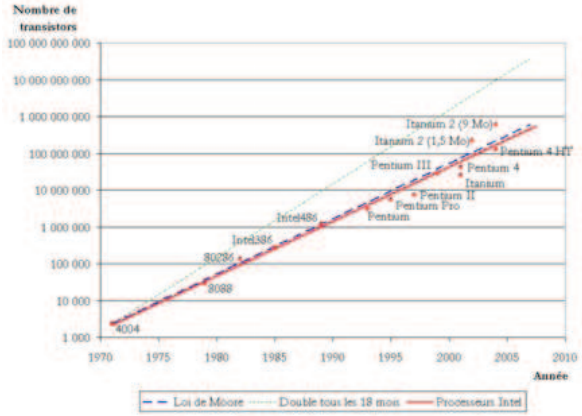
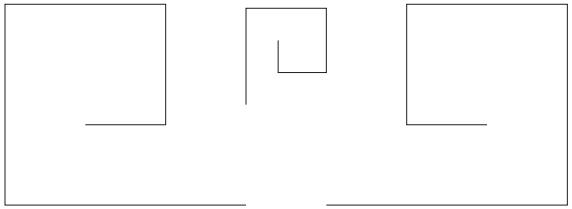


Fig. 1.19 : Spirales rectangulaires



- robustesse : cet algorithme suppose qu'on a suffisamment de place pour tracer une spirale (le dernier côté fait 6 pas de long) ; s'il fonctionne correctement sur une plage, il ne fonctionnera certainement plus dans un placard.
- réutilisabilité : il existe une infinité de spirales rectangulaires qui ont les caractéristiques attendues ; il suffit de penser à changer l'orientation initiale ou la longueur du pas par exemple. On ne pourra donc pas utiliser l'algorithme tel quel dans toutes les configurations : il aurait fallu paramétrer l'angle de rotation et la longueur du pas.
- complexité : on peut la caractériser par le nombre de pas : $2 + 3 + 4 + 5 + 6 = 20$ pas.
- efficacité : si la complexité se calcule en nombre de pas comme ci-dessus, on pourrait imaginer par exemple que la fréquence des pas soit plus grande (« fréquence d'horloge ») ou que 5 personnes prennent en charge chacune un côté de la spirale pour gagner du temps (« système multi-processeurs »).

TD 1.24 : Tracés de polygones réguliers.

1. Pentagone régulier de 10 pas de côté.
   (a) avance de 10 pas,
   (b) tourne à gauche de (360/5)°,
   (c) avance de 10 pas,
   (d) tourne à gauche de (360/5)°,
   (e) avance de 10 pas,
   (f) tourne à gauche de (360/5)°,
   (g) avance de 10 pas,
   (h) tourne à gauche de (360/5)°,
   (i) avance de 10 pas,
   (j) tourne à gauche de (360/5)°.
2. Hexagone régulier de 10 pas de côté.
   Répète 6 fois de suite les 2 instructions
   (a) avance de 10 pas,
   (b) tourne à gauche de (360/6)°,
3. Octogone régulier de 10 pas de côté.
   Répète 8 fois de suite les 2 instructions

On remarque qu'on a effectué 5 fois de suite les 2 instructions suivantes :

(a) avance de 10 pas,
(b) tourne à gauche de (360/5)°.

Pour simplifier, on écrira plutôt :
Répète 5 fois de suite les 2 instructions

(a) avance de 10 pas,
(b) tourne à gauche de (360/5)°.

C'est ce que nous ferons dans les exemples suivants.

(a) avance de 10 pas,

(b) tourne à gauche de $(360/8)°$,

4. Polygone régulier de n côtés de 10 pas chacun.
Répète n fois de suite les 2 instructions

(a) avance de 10 pas,

(b) tourne à gauche de $(360/n)°$,

TD 1.25 : Multiplication « à la russe ».

1.

multiplicande $M \times 2$	multiplicateur $m \div 2$	reste $m \bmod 2$	somme partielle
96	64	0	$(0 \times 96) + 0$
192	32	0	$(0 \times 192) + 0$
384	16	0	$(1 \times 384) + 0$
768	8	0	$(0 \times 768) + 0$
1536	4	0	$(0 \times 1536) + 0$
3072	2	0	$(0 \times 3072) + 0$
6144	1	1	$(1 \times 6144) + 0$
		$64 \times 96 =$	6144

2.

multiplicande $M \times 2$	multiplicateur $m \div 2$	reste $m \bmod 2$	somme partielle
239	45	1	$(1 \times 239) + 0$
478	22	0	$(0 \times 478) + 239$
956	11	1	$(1 \times 956) + 239$
1912	5	1	$(1 \times 1912) + 1195$
3824	2	0	$(0 \times 3824) + 3107$
7648	1	1	$(1 \times 7648) + 3107$
		$45 \times 239 =$	10755

TD 1.26 : Multiplication arabe : $63247 \times 124 = 7842628$.

Remarque 1.14 : *Boulier chinois : division par 3.*

1. « san-yi sanshi-yi » *: trois-un ? trente et un !*
   (on pose 3 à la place du 1, et on ajoute 1 à droite)
2. « san-er liushi-er » *: trois-deux ? soixante deux !*
   (on pose 6 à la place du 2, et on ajoute 2 à droite)
3. « feng san jin yi-shi » *: trois-trois ? dizaine montée !*
   (on pose 0 à la place du 3, et on ajoute 1 à gauche)

Effectuer la division $2271 \div 3$ *avec le boulier chinois.*

Remarque 1.15 : *Un entier positif en base b est représenté par une suite de chiffres* $(r_n r_{n-1} \dots r_1 r_0)_b$ *où les* r_i *sont des chiffres de la base b (*$0 \leq r_i < b$*). Ce nombre a pour valeur :*

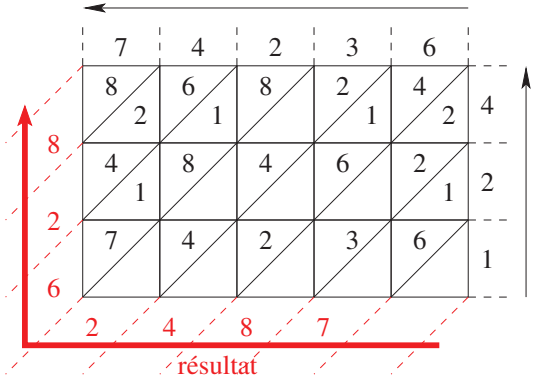
$$r_n b^n + r_{n-1} b^{n-1} + \dots + r_1 b^1 + r_0 b^0 = \sum_{i=0}^{i=n} r_i b^i$$

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 :*

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

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

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



TD 1.27 : Division chinoise : $1234 \div 7$ ($1234 = 176 \times 7 + 2$).

			↓							↓								↓						
1		0	0	0	0	→ 7-1 →	1		0	1	0	0	→ 7-5 →	1		0	1	0	0					
2		1	2	3	4		2		1	0	3	4		2		1	2	4	4					
				↓									↓											
1		0	1	0	0	→ 7-4 →	1		0	1	1	1	→ 7-7 →	1		0	1	1	0					
2		1	2	4	4		2		1	2	0	4		2		1	2	1	2					

TD 1.28 : Calcul Shadok.

Le système Shadok est un système de numération en base 4 :
GA = 0, BU = 1, ZO = 2 et MEU = 3.

1. (a) GA GA $= (00)_4 = 0$
   (b) BU BU BU $= (111)_4 = 1 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 16 + 4 + 1 = 21$
   (c) ZO ZO ZO ZO $= (2222)_4 = 2 \cdot 4^3 + 2 \cdot 4^2 + 2 \cdot 4^1 + 2 \cdot 4^0 = 128 + 32 + 8 + 2 = 170$
   (d) MEU MEU MEU MEU MEU $= (33333)_4 = 3 \cdot 4^4 + 3 \cdot 4^3 + 3 \cdot 4^2 + 3 \cdot 4^1 + 3 \cdot 4^0 = 768 + 192 + 48 + 12 + 3 = 1023$
2. (a) ZO ZO MEU + BU GA MEU $= (223)_4 + (103)_4 = (332)_4 = 43 + 19 = 62$
   (b) MEU GA MEU − BU MEU GA $= (303)_4 - (130)_4 = (113)_4 = 51 - 28 = 23$
   (c) ZO MEU MEU × BU GA MEU $= (233)_4 \times (103)_4 = (31331)_4 = 47 \times 19 = 893$
   (d) ZO ZO ZO MEU ÷ BU GA ZO $= (2223)_4 \div (102)_4 = (21)_4 = 171 \div 18 = 9$

1.6 Annexes

1.6.1 Lettre de Jacques Perret

Au printemps de 1955, IBM France s'apprêtait à construire dans ses ateliers de Corbeil-Essonnes (consacrés jusque-là au montage des machines mécanographiques — tabulatrices, trieuses, ...— de technologie électromécanique) les premières machines électroniques destinées au traitement de l'information. Aux États-Unis ces nouvelles machines étaient désignées sous le vocable Electronic Data Processing System. Le mot « computer » était plutôt réservé aux machines scientifiques et se traduisait aisément en « calculateur » ou « calculatrice ». Sollicité par la direction de l'usine de Corbeil-Essonnes, François Girard, alors responsable du service promotion générale publicité, décida de consulter un de ses anciens maîtres, Jacques Perret, professeur de philologie latine à la Sorbonne. A cet effet il écrit une lettre à la signature de C. de Waldner, président d'IBM France. Il décrit sommairement la nature et les fonctions des nouvelles machines. Il accompagne sa lettre de brochures illustrant les machines mécanographiques. Le 16 avril, le professeur Perret lui répond. L'ordinateur IBM 650 peut commencer sa carrière. Protégé pendant quelques mois par IBM France, le mot fut rapidement adopté par un public de spécialistes, de chefs d'entreprises et par l'administration. IBM décida de le laisser dans le domaine public (d'après le site de la $10^{ème}$ semaine de la langue française et de la francophonie www.semainelf.culture.fr/site2005/dixmots).

Le 16 IV 1955

Cher Monsieur,

Que diriez-vous d'« ordinateur » ? C'est un mot correctement formé, qui se trouve même dans le Littré comme adjectif désignant Dieu qui met de l'ordre dans le monde. Un mot de ce genre a l'avantage de donner aisément un verbe « ordiner », un nom d'action « ordination ». L'inconvénient est que « ordination » désigne une cérémonie religieuse ; mais les deux champs de signification (religion et comptabilité) sont si éloignés et la cérémonie d'ordination connue, je crois, de si peu de personnes que l'inconvénient est peut-être mineur. D'ailleurs votre machine serait « ordinateur » (et non ordination) et ce mot est tout à fait sorti de l'usage théologique. « Systémateur » serait un néologisme, mais qui ne me paraît pas offensant ; il permet « systématisé » ; - mais « système » ne me semble guère utilisable - « combinateur » a l'inconvénient du sens péjoratif de « combine » ; « combiner » est usuel donc peu capable de devenir technique ; « combination » ne me paraît guère viable à cause de la proximité de « combinaison ». Mais les Allemands ont bien leurs « combinats » (sorte de trusts, je crois), si bien

Fig. 1.20 : Le premier ordinateur (1946)
ENIAC (Electronic Numerical Integrator Analyser and Computer).

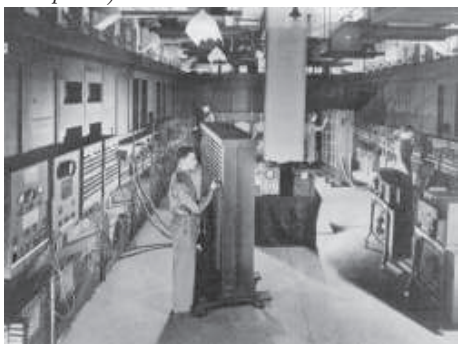


Fig. 1.21 : Premiers micro-ordinateurs

Apple II (1977)

ZX 81 (1981)

IBM PC (1981)

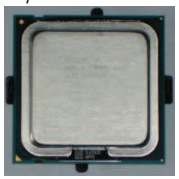
Macintosh (1984)

Fig. 1.22 : Du 8086 (1978) au Core 2 (2006)

8086

80486

Pentium 4



Core Duo

Fig. 1.23 : Micro-ordinateurs récents

PDA

Tablette

iPhone

Wii

que le mot aurait peut-être des possibilités autres que celles qu'évoque « combine ».

« Congesteur », « digesteur » évoquent trop « congestion » et « digestion ». « Synthétiseur » ne me paraît pas un mot assez neuf pour designer un objet spécifique, déterminé comme votre machine. En relisant les brochures que vous m'avez données, je vois que plusieurs de vos appareils sont désignés par des noms d'agent féminins (trieuse, tabulatrice). « Ordinatrice » serait parfaitement possible et aurait même l'avantage de séparer plus encore votre machine du vocabulaire de la théologie. Il y a possibilité aussi d'ajouter à un nom d'agent un complément : « ordinatrice d'éléments complexes » ou un élément de composition, par ex. : « sélecto-systématateur ». - « Sélecto- ordinateur » a l'inconvénient de 2 « o » en hiatus, comme « électro-ordinatrice ». Il me semble que je pencherais pour « ordinatrice électronique ». Je souhaite que ces suggestions stimulent, orientent vos propres facultés d'invention. N'hésitez pas à me donner un coup de téléphone si vous avez une idée qui vous paraisse requérir l'avis d'un philologue.

Vôtre.

J. Perret