Gilles Dowek
Jean-Pierre Archambault, Emmanuel Baccelli, Claudio Cimelli,
Albert Cohen, Christine Eisenbeis, Thierry Viéville et Benjamin Wack
Avec la contribution de Hugues Bersini et de Guillaume Le Blanc
Préface de Gérard Berry, professeur au Collège de France

Informatique et sciences du numérique

Édition spéciale Python !

Manuel de spécialité ISN en terminale

Avec des exercices corrigés
et des idées de projets

EYROLLES

, ISBN : 978-2-212-13676-0

Préface

L'année 2012 a vu l'entrée de l'informatique en tant qu'enseignement de spécialité en classe de terminale scientifique. Cette entrée devenait urgente, car l'informatique est désormais partout. Créée dans les années 1950 grâce à une collaboration entre électroniciens, mathématiciens et logiciens (ces derniers en ayant posé les bases dès 1935), elle n'a cessé d'accélérer son mouvement depuis, envahissant successivement l'industrie, les télécommunications, les transports, le commerce, l'administration, la diffusion des connaissances, les loisirs, et maintenant les sciences, la médecine et l'art, tout cela en créant de nouvelles formes de communication et de relations sociales. Les objets informatiques sont maintenant par milliards et de toutes tailles, allant du giga-ordinateur équipé de centaines de milliers de processeurs aux micro-puces des cartes bancaires ou des prothèses cardiaques et auditives, en passant par les PC, les tablettes et smartphones, les appareils photos, ou encore les ordinateurs qui conduisent et contrôlent les trains, les avions et bientôt les voitures. Tous fonctionnent grâce à la conjonction de puces électroniques et de logiciels, objets immatériels qui décrivent très précisément ce que vont faire ces appareils électroniques. Au XXI^{e} siècle, la maîtrise du traitement de l'information est devenue aussi importante que celle de l'énergie dans les siècles précédents, et l'informatique au sens large est devenue un des plus grands bassins d'emploi à travers le monde. Cela implique que de nombreux lycéens actuels participeront à son essor dans l'avenir.

Ces jeunes lycéens sont bien sûr très familiers avec les appareils informatisés. Mais ce n'est pas pour cela qu'ils en comprennent le fonctionnement, même sur des plans élémentaires pour certains. Une opinion encore fort répandue est qu'il n'y a pas besoin de comprendre ce fonctionnement, et qu'il suffit d'apprendre l'usage des appareils et

logiciels. À l'analyse, cette opinion apparemment naturelle s'avère tout à fait simpliste, avec des conséquences néfastes qu'il faut étudier de près. Pour faire un parallèle avec une autre discipline, on enseigne la physique car elle est indispensable à la compréhension de la nature de façon générale, et aussi de façon plus spécifique au travail de tout ingénieur et de tout scientifique, c'est-à-dire aux débouchés naturels de beaucoup d'élèves de terminale scientifique. Mais qui penserait qu'il suffit de passer le permis de conduire pour comprendre la physique d'un moteur ou la mécanique une voiture ? Or, nous sommes tous autant confrontés à l'informatique qu'à la physique, même si elle ne représente pas un phénomène naturel préexistant ; comme pour la physique, les ingénieurs et scientifiques devront y être au moins autant créateurs que consommateurs. Pour être plus précis, sous peine de ne rester que des consommateurs serviles de ce qui se crée ailleurs, il est indispensable pour notre futur de former au cœur conceptuel et technique de l'informatique tout élève dont le travail technique sera relié à l'utilisation avancée ou à la création de l'informatique du présent ou du futur. Il est donc bien naturel que la nouvelle formation à l'informatique s'inaugure en terminale scientifique. Mais elle devra immanquablement ensuite être élargie à d'autres classes, car tout élève sera concerné en tant que futur citoyen.

Pour être efficace, toute formation scolaire demande un support adéquat. Ce premier livre va jouer ce rôle pour l'informatique, en présentant de façon pédagogique les quatre composantes scientifiques et techniques centrales de son cœur scientifique et technique : langages de programmation, numérisation de l'information, machines et réseaux, et algorithmes. Il a été écrit par des chercheurs et enseignants confirmés, tous profondément intéressés par le fait que les élèves comprennent, assimilent et apprécient les concepts et techniques présentées. Il insiste bien sur deux points essentiels : le fait que ces quatre composantes sont tout à fait génériques, c'est-à-dire valables pour tous les types d'applications, des méga-calculs nécessaires pour étudier l'évolution du climat aux calculs légers et rapides à effectuer dans les micro-puces enfouies partout, et le fait que les concepts associés resteront valables dans le temps. En effet, si les applications de l'informatique évoluent très vite, son cœur conceptuel reste très stable, au moins au niveau approprié pour la terminale scientifique. L'enseigner de façon adéquate est nécessaire autant à la compréhension des bases qu'à tout approfondissement ultérieur. À n'en pas douter, cet ouvrage y contribuera.

Gérard Berry, directeur de recherche Inria
Professeur au Collège de France,
Membre de l'Académie des sciences, de l'Académie des technologies,
et de l'Academia Europaea

Table des matières

Avant-propos

Il y a un siècle, il n'y avait pas d'ordinateurs ; aujourd'hui, il y en a plusieurs milliards. Ces ordinateurs et autres *machines* numériques que sont les réseaux, les téléphones, les télévisions, les baladeurs, les appareils photos, les robots, etc. ont changé la manière dont nous :

- concevons et fabriquons des objets,
- échangeons des informations entre personnes,
- gardons trace de notre passé,
- accédons à la connaissance,
- faisons de la science,
- créons et diffusons des œuvres d'art,
- organisons les entreprises,
- administrons les états,
- etc.

Si les ordinateurs ont tout transformé, c'est parce qu'ils sont polyvalents, ils permettent de traiter des *informations* de manières très diverses. C'est en effet le même objet qui permet d'utiliser des logiciels de conception assistée par ordinateur, des machines à commande numérique, des logiciels de modélisation et de simulation, des encyclopédies, des cours en ligne, des bases de données, des blogs, des forums, des logiciels de courrier électronique et de messagerie instantanée, des logiciels d'échange de fichiers, des logiciels de lecture de vidéos et musique, des tables de mixage numériques, des archives numériques, etc.

Cette polyvalence s'illustre aussi par le nombre d'outils que les ordinateurs ont remplacé : machines à écrire, téléphones, machines à calculer, télévisions, appareils photos, électrophones, métiers à tisser…

En fait, les ordinateurs sont non seulement capables de traiter des informations de manières diverses, mais également de toutes les manières possibles. Ce sont des machines universelles.

Un procédé systématique qui permet de traiter des informations s'appelle un *algorithme*. Ainsi, on peut parler d'algorithmes de recherche d'un mot dans un dictionnaire, d'algorithmes de chiffrement et de déchiffrement, d'algorithmes pour effectuer des additions et des multiplications, etc.

Traiter des informations

Traiter des informations signifie appliquer, d'une manière systématique, des opérations à des symboles. La recherche d'un mot dans un dictionnaire, le chiffrement et le déchiffrement d'un message secret, l'addition et la multiplication de deux nombres, la fabrication des emplois du temps des élèves d'un lycée ou des pilotes d'une compagnie aérienne, le calcul de l'aire d'une parcelle agricole ou encore le compte des points des levées d'un joueur au Tarot sont des exemples de traitements d'informations.

ALLER PLUS LOIN **Des algorithmes aussi vieux que l'écriture**

Il y a quatre mille ans, les scribes et les arpenteurs, en Mésopotamie et en Égypte, mettaient déjà en œuvre des algorithmes pour effectuer des opérations comptables et des calculs d'aires de parcelles agricoles. La conception d'algorithmes de traitement de l'information semble remonter aux origines mêmes de l'écriture. Dès l'apparition des premiers signes écrits, les hommes ont imaginé des algorithmes pour les transformer.

De manière plus générale, un algorithme est un procédé systématique qui permet de faire quelque chose. Par exemple une recette de cuisine est un algorithme. Ainsi, même avant l'invention de l'écriture, les hommes ont conçu et appris des algorithmes, pour fabriquer des objets en céramique, tisser des étoffes, nouer des cordages ou, simplement, préparer des aliments.

Le bouleversement survenu au milieu du XX^{e} siècle tient à ce que les hommes ont cessé d'utiliser exclusivement ces algorithmes à la main ; ils ont commencé à les faire exécuter par des machines, les ordinateurs. Pour y parvenir, il a fallu exprimer ces algorithmes dans des *langages* de programmation, accessibles aux ordinateurs. Ces langages sont différents des langues humaines en ce qu'ils permettent la communication non pas entre les êtres humains, mais entre les êtres humains et les machines.

L'informatique est donc née de la rencontre de quatre concepts très anciens :

- machine,
- information,
- algorithme,
- langage.

Ces concepts existaient tous avant la naissance de l'informatique, mais l'informatique les a profondément renouvelés et articulés en une science cohérente.

Structure de l'ouvrage

L'objectif de ce cours est d'introduire les quatre concepts de machine, d'information, d'algorithme et de langage, mais surtout de montrer la manière dont ils fonctionnent ensemble. Quand nous étudierons les algorithmes fondamentaux, nous les exprimerons souvent dans un langage de programmation. Quand nous étudierons l'organisation des machines, nous verrons comment elles permettent d'exécuter des programmes exprimés dans un langage de programmation. Quand nous étudierons la notion d'information, nous verrons des algorithmes de compression, de chiffrement, etc.

Ce livre est donc organisé en quatre parties regroupant vingt-deux chapitres, dont certains d'un niveau plus avancé (indiqués par un astérisque) :

- Dans la **première partie** « **Langages** », nous apprendrons à écrire des programmes. Pour cela, nous allons découvrir les ingrédients dont les programmes sont constitués : l'affectation, la séquence et le test (**chapitre 1**), les boucles (**chapitre 2**), les types (**chapitre 3**), les fonctions (**chapitre 4***) et les fonctions récursives (**chapitre 5***). Pour finir, nous nous pencherons sur la notion de langage formel (**chapitre 6***). Dès que l'on commence à maîtriser ces concepts, il devient possible de créer ses propres programmes.
- Dans la **deuxième partie**, « **Informations** », nous abordons l'une des problématiques centrales de l'informatique : représenter les informations que l'on veut communiquer, stocker et transformer. Nous apprendrons à représenter les nombres entiers et les nombres à virgule (**chapitre 7**), les caractères et les textes (**chapitre 8**), les images et les sons (**chapitre 9**). La notion de valeur booléenne, ou de bit, qui apparaît dans ces trois chapitres, nous mènera naturellement à la notion de fonction booléenne (**chapitre 10**). Nous apprendrons ensuite à structurer de grandes quantités d'informations (**chapitre 11***), à optimiser la place occupée grâce à la compression, corriger les erreurs qui peuvent se produire au moment de la transmission et du stockage de ces informations, et à les protéger par le chiffrement (**chapitre 12***).
- Dans la **troisième partie**, « **Machines** », nous verrons que derrière les informations, il y a toujours des objets matériels : ordinateurs, réseaux, robots, etc. Les premiers ingrédients de ces machines sont des portes booléennes (**chapitre 13**) qui réalisent les fonctions booléennes vues au chapitre 10. Ces portes demandent à être complétées par d'autres circuits, comme les mémoires et les horloges, qui introduisent une dimension temporelle (**chapitre 14**). Nous découvrirons comment fonctionnent les machines que nous utilisons tous les jours (**chapitre 15**). Nous verrons que les réseaux, comme les

oignons, s'organisent en couches (**chapitre 16***). Et nous découvrirons enfin les entrailles des robots, que nous apprendrons à commander (**chapitre 17***).

- Dans la **quatrième partie,** « **Algorithmes** », nous apprendrons quelques-uns des savoir-faire les plus utiles au XXI^{e} siècle : ajouter des nombres exprimés en base deux (**chapitre 18**), dessiner (**chapitre 19**), retrouver une information par dichotomie (**chapitre 20***), trier des informations (**chapitre 21***) et parcourir un graphe (**chapitre 22***).

REMARQUE **Chapitres élémentaires et chapitres avancés***

Les chapitres avancés sont notés ici d'un astérisque. Il s'agit des deux ou trois derniers chapitres de chaque partie. Ils sont signalés en début de chapitre.

Chaque chapitre contient trois types de contenus :

- une partie de **cours** ;
- des sections intitulées « **Savoir-faire** », qui permettent d'acquérir les capacités essentielles ;
- des **exercices,** avec leur corrigé lorsque nécessaire.

 **Exercices difficiles**

Les exercices notés d'un cactus sont d'un niveau plus difficile.

Des encadrés « **Aller plus loin** » donnent des ouvertures vers des questions hors-programme. Chaque chapitre se conclut par trois questions de cours sous forme d'encadré intitulé « **Ai-je bien compris ?** ».

Les propositions de projets sont regroupées en fin de manuel.

Parcours possibles

Cet ouvrage peut être parcouru de plusieurs manières. Nous proposons par exemple de commencer par les chapitres élémentaires de la partie **Informations** (7, 8, 9 et 10), de poursuivre par ceux de la partie **Langages** (1, 2 et 3), de continuer par les chapitres avancés de la partie **Informations** (11 et 12), les chapitres élémentaires de la partie **Algorithmes** (18 et 19) et de la partie **Machines** (13, 14 et 15), et enfin de passer aux chapitres avancés de la partie **Machines** (16 et 17), de la partie **Langages** (4, 5 et 6) et de la partie **Algorithmes** (20, 21 et 22).

Il n'est pas nécessaire de lire ces chapitres au même degré de détails. À chaque élève de choisir les thématiques qu'il souhaite approfondir parmi celles proposées, en particulier par le choix de ses projets.

La seule contrainte est d'acquérir assez tôt les bases des langages de programmation, aux chapitres 1, 2 et 3, pour pouvoir écrire soi-même des programmes. Quand on apprend l'informatique, il est en effet important non seulement d'écouter des cours et de lire des livres, mais aussi de mettre en pratique les connaissances que l'on acquiert en écrivant soi-même des programmes, en se trompant et en corrigeant ses erreurs.

Remerciements

Les auteurs tiennent à remercier Ali Assaf, Olivier Billet, Manuel Bricard, Stéphane Bortzmeyer, Alain Busser, David Cachera, Vint Cerf, Julien Cervelle, Sébastien Chapuis, Arthur Charguéraud, Sylvain Conchon, S. Barry Cooper, Ariane Delrocq, Lena Domröse, Raffi Enficiaud, Jean-Christophe Filliâtre, Monique Grandbastien, Fabrice Le Fessant, Philippe Lucaud, Pierre-Étienne Moreau, Claudine Noblet, François Périnet, Gordon Plotkin, François Pottier, David Roche, Laurent Sartre, Dana Scott, Adi Shamir, Joseph Sifakis et Gérard Swinnen pour leur aide au cours de la rédaction de ce livre ainsi que l'équipe des éditions Eyrolles, Anne Bougnoux, Laurène Gibaud, Muriel Shan Sei Fan, Gaël Thomas et Camille Vorng pour leur travail éditorial très créatif et Hugues Bersini et Guillaume Le Blanc qui ont permis à ce manuel d'exister dans cette version qui utilise le langage Python.

Merci également à Christine Paulin, Raphaël Hertzog, Pierre Néron, Grégoire Péan, Jonathan Protzenko et Dominique Quatravaux pour leurs témoignages vivants.

```
getNombreAleatoire()
{
    return 4;   // Nombre choisi au dé (non truqué)
                // Garanti 100% aléatoire.
}
```

XKC

Première partie

Langages

Dans cette première partie, nous apprenons à écrire des programmes. Pour cela, nous découvrons les ingrédients dont les programmes sont constitués : l'affectation, la séquence et le test (chapitre 1), les boucles (chapitre 2), les types (chapitre 3), les fonctions (chapitre 4*) et les fonctions récursives (chapitre 5*). Pour finir, nous nous penchons sur la notion de langage formel (chapitre 6*).

Dès que l'on commence à maîtriser ces concepts, il devient possible de créer ses propres programmes.

1

John Backus (1924-2007) est l'auteur de l'un des premiers langages de programmation : le langage Fortran (1954). Il a par la suite proposé, avec Peter Naur, la *notation de Backus et Naur* qui permet de décrire des grammaires, en particulier celles des langages de programmation (voir le chapitre 6).

Grace Hopper (1906-1992) est, elle aussi, l'auteur d'un des premiers langages de programmation : le langage Cobol (1959). Avant cela, elle a été l'une des premières à programmer le Harvard Mark I de Howard Aiken, l'un des tous premiers calculateurs électroniques.

Les ingrédients des programmes

Un ordinateur peut faire bien des choses,
mais il faut d'abord les lui expliquer.

Apprendre la programmation, ce n'est pas seulement apprendre à écrire un programme, c'est aussi comprendre de quoi il est fait, comment il est fait et ce qu'il fait. Un programme est essentiellement constitué d'expressions et d'instructions. Nous introduisons dans ce premier chapitre les trois instructions fondamentales que sont l'affectation de variables, la séquence et le test.

Nous étudions les instructions en observant les transformations qu'elles opèrent sur l'état de l'exécution du programme, c'est-à-dire sur l'ensemble des boîtes pouvant contenir des valeurs, avec leur nom.

2

Gilles Kahn (1946-2006) et **Gordon Plotkin** (1946-) ont proposé des outils pour décrire la *sémantique* des langages de programmation, c'est-à-dire ce qu'il se passe quand on exécute un programme. Gilles Kahn est aussi l'auteur, avec Gérard Huet, du système *Mentor*, l'un des premiers systèmes qui permet de définir de manière complètement formelle des langages de programmation. On doit à Gordon Plotkin des contributions à de nombreux domaines de l'informatique, en particulier la démonstration automatique et la théorie des systèmes concurrents.

Les boucles

Un ordinateur est fait pour effectuer
des calculs longs et répétitifs.

Dans ce chapitre, nous introduisons une nouvelle instruction, la boucle, qui permet d'exécuter une instruction plusieurs fois. Nous en présentons deux variantes, la boucle `for` et la boucle `while`. Nous expliquons pourquoi il peut arriver que l'exécution d'une boucle ne s'arrête jamais.

3

Robin Milner (1934-2010) est l'auteur de l'un des premiers langages de programmation avec des types polymorphes et implicites : le langage ML. Ce langage est l'ancêtre de nombreux langages contemporains en particulier des langages Caml et Haskell. Par la suite il a développé l'un des premiers langages permettant de décrire des systèmes concurrents, c'est-à-dire formés de plusieurs processus qui s'exécutent en parallèle. Dans son discours de réception du prix Turing, il a insisté sur l'autonomie de l'informatique, qui n'est une partie d'aucune autre science.

Les types

Une boîte ne sait pas comment elle s'appelle. C'est à nous de lui donner un nom, et une forme.

Dans ce chapitre, nous voyons qu'il y a différents *types* de boîtes. Chaque boîte peut contenir un nombre entier, un nombre à virgule, une valeur booléenne, une chaîne de caractères… ou plusieurs, dans le cas de types composites tels les listes.

4

John McCarthy (1927-2011) est l'auteur du langage Lisp (1958), dont la principale construction est la définition de fonctions. Il est aussi l'un des inventeurs de la notion de temps partagé, qui permet à plusieurs personnes d'utiliser un même ordinateur en même temps. Il a écrit l'un des premiers programmes jouant aux échecs et inventé pour cela un algorithme, *la méthode alpha-bêta*, qui permet de jouer non seulement aux échecs, mais aussi à de nombreux autres jeux. Il a aussi été un défenseur de l'idée de progrès et de l'importance des mathématiques dans l'éducation.

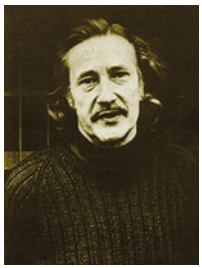
Les fonctions

Chapitre avancé

Pour avoir du style, il faut éviter les redites.

Dans ce chapitre, nous introduisons une nouvelle construction : la définition de fonction, qui permet d'isoler une instruction qui revient plusieurs fois dans un programme. Une fonction est définie par un nom, par ses arguments qui porteront les valeurs communiquées par le programme principal à la fonction au moment de son appel et éventuellement une valeur de retour communiquée au programme par la fonction en fin d'exécution.

5



Christopher Strachey (1916-1975) et **Dana Scott** (1932-) ont donné une sémantique aux définitions récursives : la définition $f = G(f)$ n'est pas circulaire, mais une définition de f comme point fixe de G. Christopher Strachey est aussi l'auteur d'un des premiers programmes jouant aux dames et de l'un des premiers programmes d'informatique musicale. Avec Michael Rabin, Dana Scott a étudié les systèmes à états et transitions (voir le chapitre 22) *non déterministes*, c'est-à-dire dans lesquels plusieurs transitions sont possibles à partir d'un même état.

La récursivité

Chapitre avancé

Une définition récursive
est une définition récursive.

Dans ce chapitre, nous voyons qu'une fonction peut s'appeler elle-même. Cette construction, alternative à celle de boucle, permet d'écrire des programmes courts et élégants.

6

L'*idéographie* de **Gottlob Frege** (1848-1925) est le premier langage formel proposé pour exprimer l'ensemble des mathématiques. Ce langage a été aujourd'hui abandonné, mais il est à l'origine de la théorie des ensembles, de la logique des prédicats et, en grande partie, des langages formels utilisés en informatique, en particulier des langages de programmation. L'origine commune des mots « langage », « logique » et « logiciel » nous rappelle les liens entre la logique – l'étude du langage mathématique – et l'informatique.

La notion de langage formel

Chapitre avancé

Les langages informatiques sont presque comme les langues naturelles. Mais pas tout à fait.

Dans ce chapitre, hors programme à l'exception de la section « Redéfinir la sémantique », nous introduisons la notion de langage formel, au-delà des langages de programmation, et nous comparons ces langages aux langues naturelles. Nous prenons l'exemple du langage HTML, présenté au chapitre 8. Ceci nous permet de présenter les notions de grammaire et de sémantique.

La grammaire définit quelles sont les chaînes de caractères, par exemple les programmes, qui sont correctement formés.

La sémantique définit ce qui se passe quand on utilise un texte, par exemple quand on exécute un programme.

XKCD

Deuxième partie

Informations

Dans cette deuxième partie, nous abordons l'une des problématiques centrales de l'informatique : représenter les informations que l'on veut communiquer, stocker et transformer. Nous apprenons à représenter les nombres entiers et les nombres à virgule (chapitre 7), les caractères et les textes (chapitre 8), et les images et les sons (chapitre 9). La notion de valeur booléenne, ou de bit, qui apparaît dans ces trois chapitres, nous mène naturellement à la notion de fonction booléenne (chapitre 10).

Nous apprenons ensuite à structurer de grandes quantités d'informations (chapitre 11*), à optimiser la place occupée grâce à la compression, corriger les erreurs qui peuvent se produire au moment de la transmission et du stockage de ces informations et à les protéger par le chiffrement (chapitre 12*).

7

Le livre de l'addition et de la soustraction d'après le calcul indien de **Muhammad al-Khwarizmi** (783 ?-850 ?), qui présente la numération décimale à position et des algorithmes permettant d'effectuer les opérations sur les nombres exprimés dans ce système, a été le vecteur de la diffusion de ce système de numération dans le bassin méditerranéen. Le mot algorithme est dérivé du nom d'al-Khwarizmi et le mot algèbre (*al-jabr*) du titre d'un autre de ses livres.

Représenter des nombres entiers et à virgule

Au commencement étaient le 0 et le 1, puis nous créâmes les nombres, les textes, les images et les sons.

Dans ce chapitre, nous voyons comment les nombres sont représentés dans les ordinateurs avec des 0 et des 1.

Nous introduisons la notion de base, en partant de la notation décimale que nous utilisons ordinairement pour écrire les nombres entiers. Nous passons par la base cinq puis décrivons la base deux, aussi appelée représentation binaire. Nous généralisons ensuite aux nombres relatifs en utilisant la notation en complément à deux, puis aux nombres à virgule, représentés par leur signe, leur mantisse et leur exposant.

8

Samuel Morse (1791-1872) est l'inventeur d'un code, dans lequel chaque lettre est exprimée par une alternance de sons brefs symbolisés par « . » et longs « – », utilisé pour télégraphier des textes. La lettre « a » y est exprimée par les sons « . – », la lettre « b » par les sons « – … », etc. Artiste peintre, Samuel Morse s'est intéressé aux télécommunications après qu'en 1925, un message lui annonçant que sa femme était malade ne lui est pas parvenu à temps. Comme nous le verrons au chapitre 12, le code morse est à références de longueurs variables, mais ce n'est pas un code préfixe.

Représenter des caractères et des textes

Les lettres ? Toutes des nombres !

Dans ce chapitre, nous voyons comment sont représentés les caractères et les textes de toutes les langues du monde.

Nous expliquons pourquoi il existe plusieurs codes tels *ASCII*, *latin-1*, *latin-2*, *UTF-32*, *UTF-8*. Nous présentons ensuite les formats enrichis qui permettent de décrire la forme des caractères et des textes, comme le font les logiciels de traitement de texte.

Un exemple de format enrichi est le langage HTML.

9

Claude Shannon (1916-2001), a montré en 1949, en s'appuyant sur des travaux antérieurs de Harry Nyquist, que la fréquence d'échantillonnage d'un son, et plus généralement d'un signal, doit être au moins le double de la fréquence maximale contenue dans ce son, pour que le son puisse être restitué à partir de l'échantillon. Il a également montré comment décrire les circuits électroniques par des fonctions booléennes (voir le chapitre 13) et comment exprimer toutes les fonctions booléennes et arithmétiques à l'aide des fonctions booléennes élémentaires (voir le chapitre 10).

Représenter des images et des sons

Pour décrire une image, commence par comprendre comment œil la voit.

C'est encore avec des 0 et des 1 que l'on représente les images et les sons, mais en grand nombre.

Pour décrire une image, on peut utiliser une représentation vectorielle en décrivant des formes géométriques : un cercle, une droite, etc., ou une représentation bitmap en quadrillant l'image et en décrivant chaque case (*pixel*). Noir et blanc, niveaux de gris ou couleurs sont décrits par différents codages.

Les sons aussi sont échantillonnés en découpant le temps durant lequel le son est émis.

Les images et les sons sont de très longues suites de 0 et de 1, nous introduisons dans ce chapitre les unités pour mesurer la taille des fichiers.

10

Dans le Manoir de Bletchey Park, quartier général des services de renseignement britanniques, **Thomas Flowers** (1905-1998) a construit pendant la seconde guerre mondiale la machine *Colossus*, premier calculateur électronique à utiliser le système binaire. Si cette machine n'était pas encore un ordinateur, elle a cassé les codes secrets (voir le chapitre 12) utilisés par l'armée allemande et a été un élément essentiel dans la victoire alliée. Plusieurs milliers de personnes ont travaillé à Bletchey Park, en particulier Alan Turing.

Les fonctions booléennes

Pour un oui… ou pour un non ? Oui ou non ?

Les fonctions booléennes sont utilisées partout : dans les langages de programmation, en architecture des ordinateurs, dans certains algorithmes cryptographiques…

Elles peuvent être décrites par des tables ou de manière symbolique et nous montrons comment passer d'une représentation à une autre.

Nous nous concentrons dans ce chapitre sur les fonctions *non*, *et* et *ou* qui permettent d'exprimer toutes les autres.

11

En 1989, **Tim Berners-Lee** (1955-) a proposé un outil aux nombreux chercheurs de l'Organisation européenne pour la recherche nucléaire (CERN) pour partager de grandes quantités d'informations : insérer dans des textes des liens vers d'autres textes, situés sur d'autres ordinateurs, auxquels on accède à travers le réseau Internet. Cette toile d'araignée de liens a vite trouvé un nom : le Web. Tim Berners Lee est l'auteur du langage HTML et du premier navigateur : Nexus.

Structurer l'information

Chapitre avancé

Comment trouver son chemin
dans une jungle d'informations ?

Dans ce chapitre, nous voyons comment les informations s'effacent, se conservent, s'organisent, selon les usages que nous voulons en faire, de la notion de fichier en arborescence et de liens à celle de base de données.

Nous insistons ici sur la notion de persistance des données avec le danger de l'hypermnésie et discutons les enjeux autour de la « gratuité » de la copie ou diffusion de l'information.

12

Ronald Rivest (1947-), **Adi Shamir** (1952-) et **Len Adleman** (1945-) ont conçu, en 1978, une méthode de chiffrement, la méthode RSA, fondée sur l'utilisation de deux clés : une clé privée et une clé publique. La méthode la plus rapide connue à ce jour pour retrouver la clé privée à partir de la clé publique est la décomposition d'un nombre entier en un produit de facteurs premiers, calcul qui demande un temps très long, quand le nombre à factoriser dépasse quelques milliers de chiffres binaires.

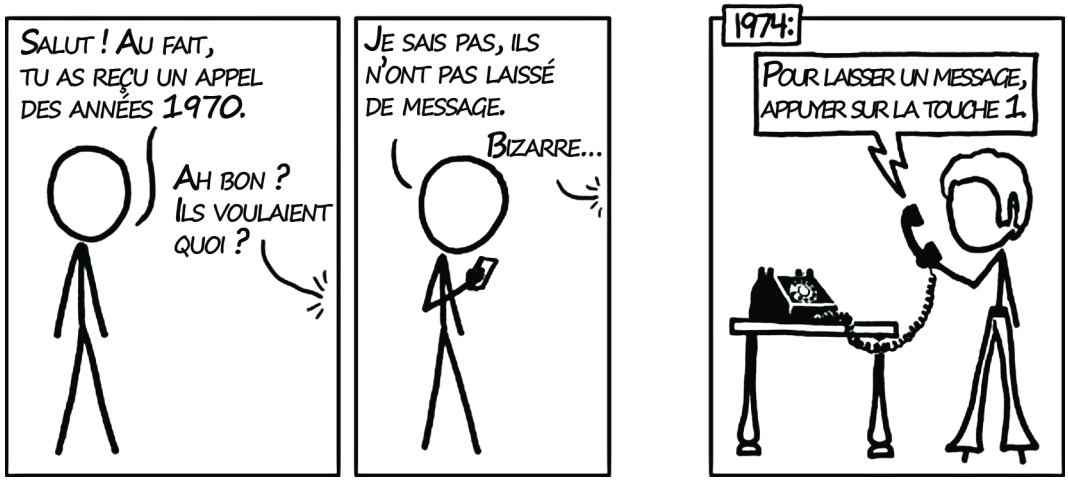
Compresser, corriger, chiffrer

Chapitre avancé

« Zhqm, zmgm, zmfm. »
(Jules César, 47 av. J.-C.)

Nous voyons maintenant comment modifier l'expression des données dans le but d'économiser de l'espace, de corriger des erreurs ou de les protéger. Pour cela, nous utilisons des formats beaucoup plus sophistiqués que ceux vus aux chapitres 8 et 9.

Pour la compression, nous expliquons comment définir un dictionnaire des mots utilisés et les coder selon leur fréquence. Pour détecter et corriger les erreurs, nous expliquons comment utiliser la redondance de l'information et exploiter des bits de contrôle de cohérence.
Pour le chiffrement, nous expliquons les méthodes à clé et introduisons les notions de *clé publique* et de *clé privée*.


SALUT ! AU FAIT, TU AS REÇU UN APPEL DES ANNÉES 1970.
AH BON ? ILS VOULAIENT QUOI ?
JE SAIS PAS, ILS N'ONT PAS LAISSÉ DE MESSAGE.
BIZARRE...
1974:
POUR LAISSER UN MESSAGE, APPUYER SUR LA TOUCHE 1.

XKCD

Troisième partie

Machines

Dans cette troisième partie, nous voyons que derrière les informations, il y a toujours des objets matériels : ordinateurs, réseaux, robots, etc. Les premiers ingrédients de ces machines sont des portes booléennes (chapitre 13) qui réalisent les fonctions booléennes vues au chapitre 10. Ces portes demandent à être complétées par d'autres circuits, comme les mémoires et les horloges, qui introduisent une dimension temporelle (chapitre 14). Nous découvrons comment fonctionnent ces machines que nous utilisons tous les jours (chapitre 15). Nous verrons que les réseaux, comme les oignons, s'organisent en couches (chapitre 16*). Et nous découvrons enfin les entrailles des robots, que nous apprenons à commander (chapitre 17*).

13

Frances Allen (1932-) est une pionnière de la parallélisation automatique des programmes, c'est-à-dire de la transformation de programmes destinés à être exécutés sur un ordinateur séquentiel – contenant un unique processeur – en des programmes destinés à être utilisés sur un ordinateur parallèle – contenant plusieurs processeurs. Elle est aussi à l'origine de nouvelles méthodes, fondées sur la théorie des graphes, pour optimiser les programmes. Elle a reçu le prix Turing en 2006 pour ces travaux.

Les portes booléennes

Au commencement était le transistor,
puis nous créâmes les portes booléennes
et, à la fin de la journée, les ordinateurs.

Dans ce chapitre, nous voyons de quoi sont faits les ordinateurs à l'échelle microscopique. Nous partons du transistor et construisons successivement des circuits *non* et *ou* qui vont nous permettre ensuite de construire les circuits de toutes les fonctions booléennes, comme nous l'avons vu au chapitre 10.

14

Otto Schmitt (1913-1998) est un pionnier du *génie biomédical*. En 1934, en étudiant la propagation de l'influx nerveux dans les nerfs des calmars, il a compris qu'un circuit en boucle fermée positive – c'est-à-dire dans lequel la sortie est connectée à l'entrée, sans inversion de valeur – avait deux états stables et pouvait donc être utilisé pour mémoriser une grandeur. En électronique, une *bascule de Schmitt* est une forme de circuit bistable, qui utilise cette idée de boucle fermée positive.

Le temps et la mémoire

Le temps est ce qui permet d'éviter
de tout faire en même temps.

Dans ce chapitre, nous voyons comment les circuits électroniques prennent le temps en compte. Nous voyons d'abord comment fabriquer un circuit mémoire. Puis, comment un circuit particulier, l'horloge, permet de synchroniser tous les autres.

15

Dans les années 1940, à l'Université de Pennsylvanie, **John Von Neumann** (1903-1957) a conçu, avec Presper Eckert et John Mauchly, deux des premiers ordinateurs : l'ENIAC, puis l'EDVAC. Ces ordinateurs étaient organisés selon l'architecture de Von Neumann, utilisée dans la quasi-totalité des ordinateurs conçus depuis : séparation du processeur et de la mémoire, reliés par un bus de communication. L'ENIAC pesait vingt-sept tonnes.

L'organisation d'un ordinateur

Pour fabriquer un ordinateur,
il suffit d'un fer à souder (ou presque…).

Dans ce chapitre, nous voyons de quoi sont faits les ordinateurs à une échelle plus proche de la nôtre et comment architectures et langages sont liés. Nous décrivons la manière dont sont assemblés le processeur de calcul, l'organisation de la mémoire et les bus permettant la circulation des données. Nous voyons comment programmer le processeur au moyen d'un langage machine simple et expliquons comment dérouler une séquence d'instructions. Nous adjoignons enfin les périphériques pour obtenir un ordinateur.


VOUS VOULEZ VOIR UNE ILLUSION D'OPTIQUE ?
TENEZ VOTRE CLAVIER FACE À VOUS, ET REGARDEZ LA TOUCHE « DÉBUT ».
MAINTENANT, LOUCHEZ UN PEU POUR QUE LE « G » ET LE « H » SE SUPERPOSENT.
MAINTENEZ VOTRE REGARD FOCALISÉ ET LEVEZ LE CLAVIER AU-DESSUS DE VOTRE TÊTE.
ARGH !
MOUHHHAHA !

XKCD

16

Vinton Cerf (1943-) et **Robert Kahn** (1938-) ont inventé, au début des années 1970, le protocole de transmission de paquets de données IP (*Internet Protocol*) et le protocole de contrôle de flux de données TCP (*Transmission Control Protocol*). Il s'agit des deux principaux protocoles du réseau Internet. Ils donnent à ce réseau sa fiabilité, sa robustesse en cas de pannes ou de modifications et sa capacité à évoluer. Cela a valu à leurs auteurs le surnom de « pères d'Internet ».

Les réseaux

Chapitre avancé

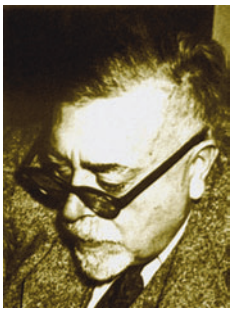
Les ordinateurs parlent aux ordinateurs.

Dans ce chapitre, nous voyons comment les ordinateurs communiquent entre eux, et comment ces communications se composent pour faire fonctionner le réseau Internet. Ces mécanismes de communication de machine à machine s'appellent des protocoles.

Les protocoles de la couche physique connectent les bus des ordinateurs. Les protocoles de la couche lien organisent un réseau local autour d'un serveur et repèrent les ordinateurs par l'adresse MAC de leur carte réseau. Les protocoles de la couche réseau organisent les réseaux locaux de proche en proche et repèrent les ordinateurs par leur adresse IP.

Nous expliquons comment les informations sont acheminées au travers du réseau à l'aide de routeurs.

17



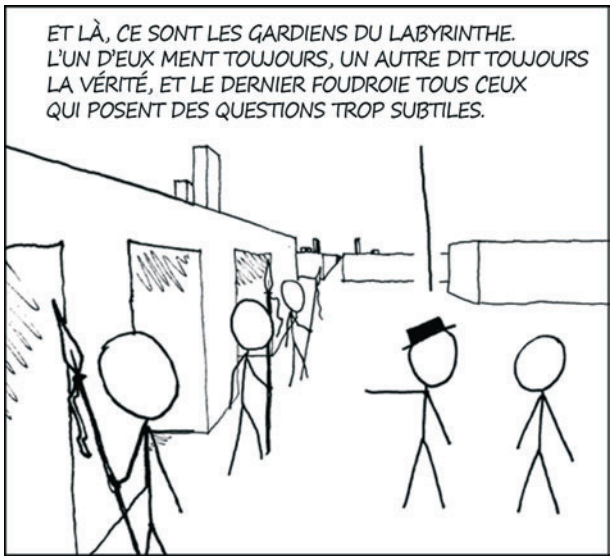
Norbert Wiener (1894-1964) est le fondateur, à la fin des années 1940, de la science du pilotage ou *cybernétique*. Entouré d'un groupe interdisciplinaire de mathématiciens, logiciens, anthropologues, psychologues, économistes…, il a cherché à comprendre les processus de commande et de communication chez les êtres vivants, dans les machines et dans les sociétés. Le concept central de la cybernétique est celui de causalité circulaire ou contrôle en boucle fermée (*feedback*).

Les robots

Chapitre avancé

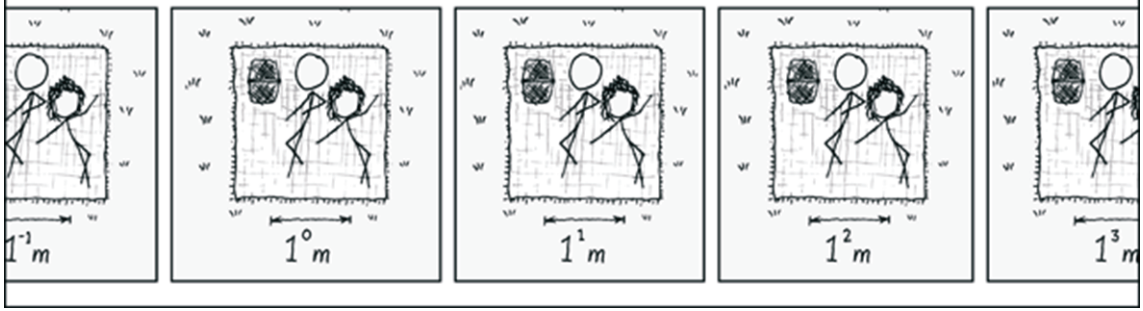
Un robot ? C'est un ordinateur à deux roues.

Dans ce chapitre, nous introduisons de nouveaux objets : les robots, qui sont essentiellement des ordinateurs munis de capteurs et d'actionneurs. Nous voyons comment les grandeurs captées sont numérisées et comment le principe de la boucle fermée permet de contrôler une action. Enfin, nous voyons comment programmer un robot à l'aide d'une boucle infinie dans laquelle les capteurs sont interrogés et les actionneurs activés.


ET LÀ, CE SONT LES GARDIENS DU LABYRINTHE.
L'UN D'EUX MENT TOUJOURS, UN AUTRE DIT TOUJOURS
LA VÉRITÉ, ET LE DERNIER FOUDROIE TOUS CEUX
QUI POSENT DES QUESTIONS TROP SUBTILES.

Puissances de Un

Une autre vision du monde.



XKCD

Quatrième partie

Algorithmes

Dans cette quatrième partie, nous apprenons quelques-uns des savoir-faire les plus utiles au XXI^{e} siècle : ajouter des nombres exprimés en base deux (chapitre 18), dessiner (chapitre 19), retrouver une information par dichotomie (chapitre 20*), trier des informations (chapitre 21*) et parcourir un graphe (chapitre 22*).

18

Ada Lovelace (1815-1852) est l'auteur du premier algorithme destiné à être exécuté par une machine. Cet algorithme, qui permettait de calculer une suite de nombres de Bernoulli, devait être exécuté sur la machine analytique conçue par Charles Babbage. Malheureusement, Babbage n'a jamais réussi à terminer sa machine. Ada Lovelace est parfois considérée comme le premier programmeur de l'histoire. Le langage de programmation Ada est ainsi nommé en son honneur.

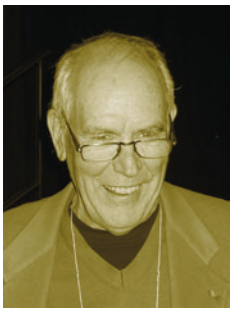
Ajouter deux nombres exprimés en base deux

Pour faire une addition, l'ordinateur fait comme on lui a appris sur les bancs de l'école.

Dans ce chapitre, nous détaillons l'algorithme de l'addition en base deux, ce qui est surtout un prétexte pour comprendre comment démontrer qu'un algorithme est correct.

L'algorithme est le même que celui que nous utilisons couramment lorsque nous effectuons une addition ordinaire, c'est-à-dire en base dix. Nous démontrons ensuite que l'algorithme que nous avons programmé calcule bien la somme de deux nombres. Pour cela, nous utilisons la notion importante d'invariant de boucle qui est une propriété vraie à chaque tour de boucle. Nous montrons une telle propriété par récurrence sur le numéro du tour de boucle.

19



Ivan Sutherland (1938-) est un des pionniers de l'informatique graphique. Il est l'auteur du logiciel *Sketchpad* (1963) qui est l'un des premiers logiciels de conception assistée par ordinateur. Ivan Sutherland a aussi été à l'origine de l'un des premiers systèmes de réalité virtuelle muni d'un visiocasque. Il est l'un des pionniers des architectures d'ordinateurs spécialisées pour le temps réel et le graphisme.

Dessiner

Ou comment devenir Botticelli
sans se tacher les doigts.

Dans ce chapitre, nous voyons comment programmer un ordinateur pour dessiner ou modifier une image… sans utiliser un logiciel de retouche photo ! Nous voyons comment ouvrir une fenêtre graphique, créer une image, dessiner en trois dimensions, lire et produire des fichiers contenant des images, transformer des images.

20

Donald Knuth (1938-) est un des fondateurs de l'algorithmique, la partie de l'informatique qui étudie les propriétés des algorithmes, indépendamment de leur expression dans un langage de programmation particulier. Le troisième volume de son livre *The art of computer programming*, intitulé *Sorting and searching*, est entièrement consacré aux algorithmes de tri et de recherche en table. Pour écrire ce livre, il a d'abord écrit un logiciel de traitement de texte : TeX, dont le nom est formé des trois premières lettres du mot *technè*, qui signifie à la fois « art » et « technique ».

La dichotomie

Chapitre avancé

Ou diviser pour régner.

Dans ce chapitre, nous voyons une méthode algorithmique générale qui permet, entre autres choses, de trouver rapidement un mot dans un dictionnaire : ouvrir le dictionnaire au milieu et s'interroger : « Le mot recherché est-il avant ou après le mot lu ? ».

Cette méthode fonctionne plus généralement dès que nous cherchons à retrouver un élément dans une liste ordonnée selon une relation d'ordre total, notion que nous définissons. Cette méthode est rapide et s'applique à de nombreux problèmes : la recherche d'un élément dans une table ordonnée, la conversion d'une valeur analogique en une valeur numérique, la recherche d'un zéro d'une fonction continue et strictement monotone, etc.

21

Philippe Flajolet (1948-2011) est un des pionniers de l'analyse de la complexité des algorithmes, c'est-à-dire du temps que dure leur exécution et de la quantité de mémoire qu'elle demande. Il a montré l'utilité, dans ce domaine, de plusieurs théories mathématiques : la combinatoire et le dénombrement, la théorie des probabilités et la théorie des fonctions d'une variable complexe. Il a aussi eu conscience très tôt de l'apport des logiciels de calcul formel pour effectuer les calculs, parfois fastidieux, que cette analyse demande.

Trier

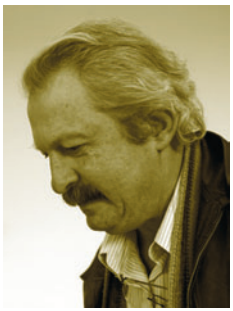
Chapitre avancé

Dans une très grande bibliothèque,
il faut classer les livres pour les retrouver.

Dans ce chapitre, nous voyons deux algorithmes de tri, ce qui est surtout un prétexte pour nous interroger sur la complexité des algorithmes.

Nous présentons le tri par sélection et le tri par fusion, et nous nous interrogeons sur l'efficacité de ces deux algorithmes, en évaluant leur temps d'exécution.

22



Joseph Sifakis (1946-) est un chercheur français d'origine grecque qui a reçu le prix Turing en 2007, avec Edmund Clarke et Allen Emerson, pour la *méthode d'énumération et de vérification des modèles*. Cette méthode se fonde sur une description des systèmes informatiques par des systèmes à états et transitions et sur une analyse des états accessibles dans ces systèmes, qui s'inspire des algorithmes de parcours de graphes. Il est le premier scientifique français à avoir reçu ce prix.

Parcourir un graphe

Chapitre avancé

Où on trouve enfin la sortie du labyrinthe.

Dans ce dernier chapitre, nous voyons un algorithme de parcours de graphe qui permet, par exemple, de chercher la sortie d'un labyrinthe, en évitant de tourner en rond en conservant à chaque étape une liste des chemins à prolonger.

Diverses variantes de cette méthode permettent de parcourir le graphe en profondeur ou en largeur.

Partant de l'exemple des labyrinthes, nous définissons la notion de graphe. Cette notion est importante car les graphes permettent de modéliser nombre de problèmes, par exemple de nombreux jeux, où les sommets représentent les états possibles du jeu et les arêtes représentent les coups possibles.

Idées de projets

Un générateur d'exercices de calcul mental

Programmer un générateur d'exercices de calcul mental : le programme choisit aléatoirement une opération et deux nombres, et vérifie la réponse de l'utilisateur. On peut ensuite poser une série de questions et compter le score total. On pourra enfin prévoir plusieurs niveaux de difficulté selon les opérations proposées ou la taille des nombres à calculer, et laisser l'utilisateur choisir son niveau de difficulté ou attribuer des scores variables aux réponses.

Mastermind

Écrire un programme qui lit deux listes de quatre éléments au clavier et indique le nombre d'éléments en commun dans ces deux listes. Écrire un programme qui joue au Mastermind : tire au hasard la combinaison secrète et répond aux propositions de l'autre joueur.

Brin d'ARN

Chercher sur le Web ce qu'est un brin d'ARN messager et comment il code pour une protéine. Écrire un programme qui détermine la protéine pour laquelle un brin d'ARN messager code.

Bataille navale

Écrire un programme de bataille navale avec plusieurs bateaux.

Cent mille milliards de poèmes

Chercher sur le Web, ou dans une bibliothèque, ce qu'est le recueil *Cent mille milliards de poèmes* de Raymond Queneau. Écrire un programme qui affiche ces cent mille milliards de poèmes.

Site de rencontres

Programmer le moteur d'un site de rencontres, sur le principe suivant :

- Chaque personne inscrite sur le site répond à un questionnaire de personnalité dont les réponses sont des entiers entre 1 et 10.
- On stocke les réponses dans une liste à double entrée : la case de la ligne `i` et de la colonne `j` contient la réponse de l'inscrit numéro `i` à la question numéro `j`.

1

John Backus (1924-2007) est l'auteur de l'un des premiers langages de programmation : le langage Fortran (1954). Il a par la suite proposé, avec Peter Naur, la *notation de Backus et Naur* qui permet de décrire des grammaires, en particulier celles des langages de programmation (voir le chapitre 6).

Grace Hopper (1906-1992) est, elle aussi, l'auteur d'un des premiers langages de programmation : le langage Cobol (1959). Avant cela, elle a été l'une des premières à programmer le Harvard Mark I de Howard Aiken, l'un des tous premiers calculateurs électroniques.

Les ingrédients des programmes

Un ordinateur peut faire bien des choses,
mais il faut d'abord les lui expliquer.

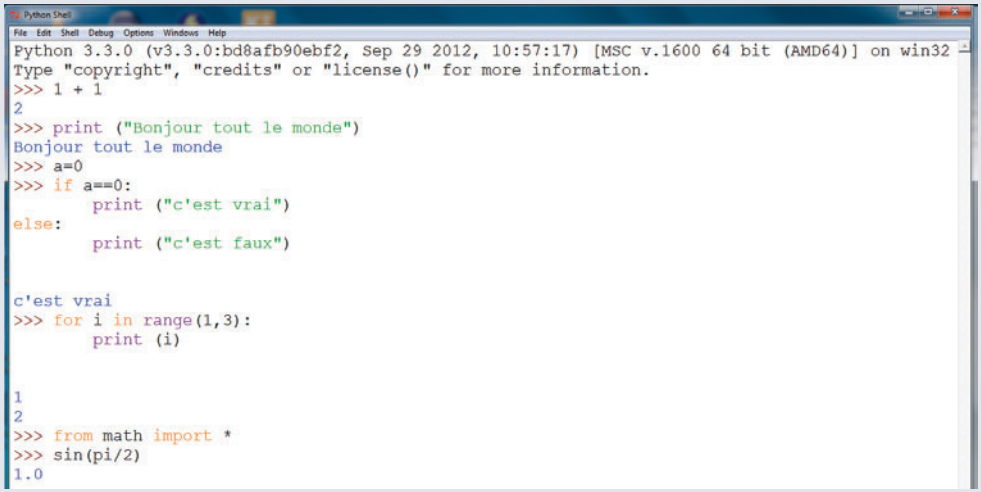
Apprendre la programmation, ce n'est pas seulement apprendre à écrire un programme, c'est aussi comprendre de quoi il est fait, comment il est fait et ce qu'il fait. Un programme est essentiellement constitué d'expressions et d'instructions. Nous introduisons dans ce premier chapitre les trois instructions fondamentales que sont l'affectation de variables, la séquence et le test.

Nous étudions les instructions en observant les transformations qu'elles opèrent sur l'état de l'exécution du programme, c'est-à-dire sur l'ensemble des boîtes pouvant contenir des valeurs, avec leur nom.

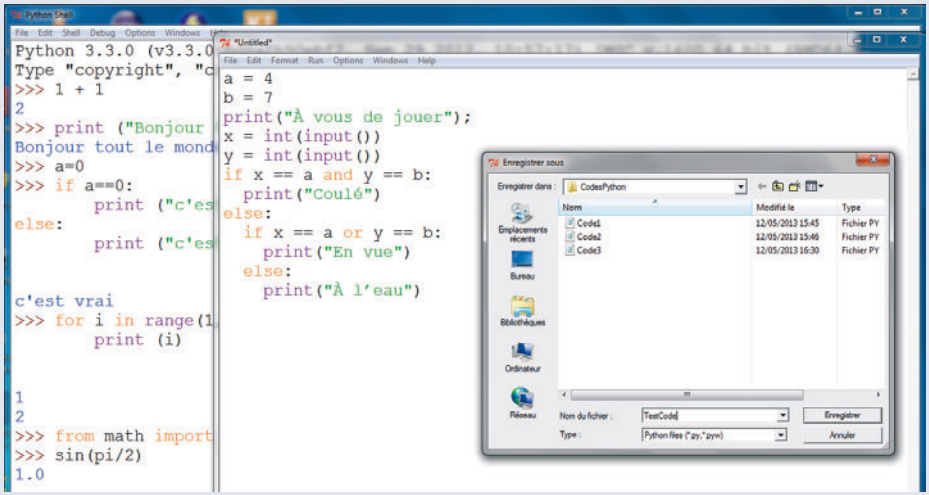
Un programme est un texte qui décrit un algorithme que l'on souhaite faire exécuter par une machine. Ce texte est écrit dans un langage particulier, appelé *langage de programmation*. Il existe plusieurs milliers de langages de programmation, parmi lesquels Python, Java, C, Caml, Fortran, Cobol, etc. Il n'est cependant pas nécessaire d'apprendre ces langages les uns après les autres, car ils sont tous plus ou moins organisés autour des mêmes notions : affectation, séquence, test, boucle, fonction, etc. Ce sont ces notions qu'il importe de comprendre. Dans ce livre, on utilise principalement le langage Python dans sa version 3. Mais rien de ce qui est dit ici n'est propre à ce langage et les élèves qui en utilisent un autre n'auront aucun mal à transposer.

EN PRATIQUE **Exécuter un programme Python**

Pour rédiger un programme, il faut lancer l'environnement de programmation IDLE de Python qui est une fenêtre permettant d'exécuter les instructions au fur et à mesure qu'on les tape.



Pour exécuter un programme plus long, tels ceux rencontrés dans ce chapitre, on peut ouvrir une fenêtre d'édition avec la commande *New window* du menu *File* et y écrire le programme, l'enregistrer dans un fichier dont le nom se termine par `.py`. et l'exécuter avec la commande *Run module* du menu *Run*. Le résultat apparaît à l'écran dans la fenêtre où l'on a exécuté les instructions précédentes.



Un premier programme

Voici un premier petit programme écrit en Python :

```
a = 4
b = 7
print("À vous de jouer")
x = int(input())
y = int(input())
if x == a and y == b:
  print("Coulé")
else:
  if x == a or y == b
    print("En vue")
  else:
    print("À l'eau")
```

Quand on exécute ce programme, il affiche `À vous de jouer` puis attend que l'on tape deux nombres au clavier. Si ces deux nombres sont 4 et 7, il affiche `Coulé` ; si le premier est 4 ou le second 7, mais pas les deux, il affiche `En vue`, sinon il affiche `À l'eau`.

DANS D'AUTRES LANGAGES Texas Instruments et Casio

Ce même algorithme peut s'exprimer dans de nombreux langages. À titre d'exemple, le voici exprimé dans le langage des calculatrices Texas Instruments et Casio. Dans ces deux cas l'*algorithme* utilisé est le même qu'en Python. Les seules différences sont dans la manière d'exprimer cet algorithme : la variable à affecter est située à droite de la flèche pour les calculatrices, l'instruction de test est structurée par des mots-clés supplémentaires, etc.

- Texas Instruments

```
PROGRAM:BATAILLE
:4 → A
:7 → B
:Disp "À vous de jouer"
:Input X
:Input Y
:If X = A et Y = B
:Then
:Disp "Coulé"
:Else
:If X = A ou Y = B
:Then
:Disp "En vue"
:Else
:Disp "À l'eau"
:End
:End
```

- Casio

```
======BATAILLE ======
4 → A
7 → B
"À vous de jouer"
? → X
? → Y
If X = A And Y = B
Then "Coulé"
Else
If X = A Or Y = B
Then "En vue"
Else "À l'eau"
IfEnd
IfEnd
```

Ce programme permet de jouer à la bataille navale, dans une variante simplifiée dans laquelle il n'y a qu'un seul bateau, toujours placé au même endroit et qui n'occupe qu'une seule case de la grille. On considère qu'un bateau est « en vue » si la case proposée est sur la même ligne ou la même colonne que le bateau.

Exercice 1.1

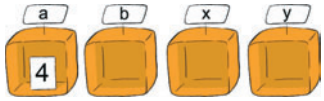
Modifier ce programme afin qu'il affiche `À toi de jouer` et non `À vous de jouer`.

Exercice 1.2

Modifier ce programme afin que le bateau soit sur la case de coordonnées `(6 ; 9)`.

La description du programme

Commençons par observer ce programme pour essayer d'en comprendre la signification. La première ligne contient l'instruction `a = 4`. Pour comprendre ce qu'il se passe quand on exécute cette instruction, il faut imaginer que la mémoire de l'ordinateur que l'on utilise est composée d'une multitude de petites boîtes. Chacune de ces boîtes porte un nom et peut contenir une valeur. Exécuter l'instruction `a = 4` a pour effet de mettre la valeur `4` dans la boîte de nom `a`.



Après avoir exécuté cette instruction, on exécute `b = 7`, ce qui a pour effet de mettre la valeur 7 dans la boîte de nom `b`.

On exécute ensuite l'instruction `print("À vous de jouer")`, ce qui a pour effet d'afficher à l'écran `À vous de jouer`.

On exécute ensuite l'instruction `x = int(input())`, ce qui a pour effet d'interrompre le déroulement du programme jusqu'à ce que l'utilisateur tape un nombre au clavier. Ce nombre est alors mis dans la boîte de nom `x`.

De même, exécuter l'instruction `y = int(input())` a pour effet d'interrompre le déroulement du programme jusqu'à ce que l'utilisateur tape un nombre au clavier. Ce nombre est alors mis dans la boîte de nom `y`. À ce point du programme, les boîtes de nom `a`, `b`, `x` et `y` contiennent chacune un nombre. Les nombres 4 et 7 pour les deux premières et les deux nombres entrés au clavier par l'utilisateur pour les deux dernières.

L'exécution du programme doit alors différer selon que les deux nombres donnés par l'utilisateur sont 4 et 7 ou non. Si c'est le cas, on veut afficher `Coulé`, si ce n'est pas le cas on veut faire autre chose. C'est ce que fait l'instruction :

```
if x == a and y == b:
  print("Coulé")
else:
  if x == a or y == b:
    print("En vue")
  else:
    print("À l'eau")
```

Quand on écrit cette instruction, il est essentiel de décaler les lignes 2, 4 et 6 de deux caractères vers la droite, et les lignes 5 et 7 de quatre caractères en commençant chaque ligne par des espaces. Cela s'appelle *indenter* cette instruction. Exécuter cette instruction a pour effet de calculer la valeur de l'expression booléenne `x == a and y == b`, où l'opération `and` est la conjonction *et*. Cette valeur est `True` (vrai) quand `x` est égal à `a` et `y` est égal à `b`, ou `False` (faux) quand ce n'est pas le cas. En fonction de la valeur de cette expression, on exécute ou bien l'instruction `print("Coulé")` ou bien l'instruction :

```
if x == a or y == b:
  print("En vue")
else:
  print("À l'eau")
```

Cette instruction étant de la même forme, son exécution a pour effet de calculer la valeur de l'expression booléenne `x == a or y == b`, où l'opération `or` est la conjonction *ou*, et en fonction de la valeur de cette expression d'exécuter ou bien l'instruction `print("En vue")` ou bien l'instruction `print("À l'eau")`.

Dans bien des cas, comme dans cet exemple, on ne veut pas simplement choisir entre deux instructions mais davantage : ici afficher `Coulé`, `En vue` ou `À l'eau`. Une possibilité est d'utiliser deux tests successifs. Une autre est d'utiliser une construction spéciale : `elif`. Ainsi, le programme ci-avant peut aussi s'écrire :

```
if x == a and y == b:
  print("Coulé")
elif x == a or y == b:
  print("En vue")
else:
  print("À l'eau")
```

Exercice 1.3

En C, le même extrait de programme s'écrit ainsi :

```
a = 4;
b = 7;
printf("À vous de jouer\n");
scanf("%d",&x);
scanf("%d",&y);
if (x == a && y == b) {
  printf("Coulé\n");}
else {
  if (x == a || y == b) {
    printf("En vue\n");}
  else {
    printf("À l'eau\n");}}
```

Quelles sont les ressemblances et les différences entre Python et C ?

SAVOIR-FAIRE Modifier un programme existant pour obtenir un résultat différent

L'intérêt de partir d'un programme existant est qu'il n'est pas toujours nécessaire d'en comprendre le fonctionnement en détail pour l'adapter à un nouveau besoin.
Il importe avant tout :

- d'identifier les parties du programme qui doivent être modifiées et celles qui sont à conserver,
- de les modifier en conséquence,
- éventuellement d'adapter les entrées et les sorties au nouveau programme,
- et, comme toujours, de le tester sur des exemples bien choisis.

Exercice 1.4 (avec corrigé)

Le programme suivant permet de calculer le prix toutes taxes comprises d'un article, connaissant son prix hors taxes, dans le cas où le taux de TVA est de 19,6 %.

```
print("Quel est le prix hors taxes ?")
ht = float(input())
ttc = ht + ht * 19.6 / 100.0
print("Le prix toutes taxes comprises est ",end="")
print(ttc)
```

Adapter ce programme pour permettre à l'utilisateur de choisir le taux de TVA.

Même si l'on n'est pas un expert en calcul de pourcentages, on identifie facilement qu'il faut remplacer le `19.6` *de la troisième ligne par un taux quelconque. Pour que ce taux puisse être*

choisi par l'utilisateur, il doit être stocké dans une nouvelle boîte : appelons-la `taux`*. Le contenu de cette boîte doit être saisi au clavier. Il faut donc prévoir l'entrée correspondante. Voici donc le nouveau programme avec, en gras, les éléments ajoutés ou modifiés :*

```
print("Quel est le prix hors taxes ?")
ht = float(input())
print(" Quel est le taux de TVA ? ")
taux = float(input())
ttc = ht + ht * taux / 100.0
print("Le prix toutes taxes comprises est ",end="")
print(ttc)
```

Exercice 1.5

En général, à la bataille navale, un bateau n'est « en vue » que si la case touchée est immédiatement voisine de celle du bateau. Modifier le premier programme de ce chapitre pour tenir compte de cette règle. On pourra traiter le cas où les cases diagonalement adjacentes au bateau sont « en vue » et le cas où elles ne le sont pas.

Les ingrédients d'un programme

Le programme de bataille navale utilise des instructions de différentes formes :

- des *affectations* de la forme `v = e` où `v` est une variable et `e` une expression,
- des *instructions d'entrée* de la forme `v = int(input())` où `v` est une variable,
- des *instructions de sortie* de la forme `print(e)` où `e` est une expression,
- des *séquences* de la forme `p q` (c'est-à-dire `p` suivi de `q`) où `p` et `q` sont deux instructions,
- des *tests* de la forme :

```
if e:
  p
else:
  q
```

où `e` est une expression et `p` et `q` deux instructions.

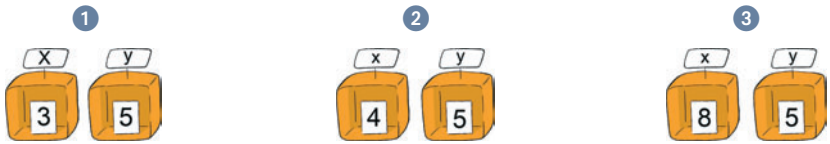
La mémoire d'un ordinateur est constituée d'une multitude de petites boîtes, un programme n'utilise en général que quelques-unes de ces boîtes. Chaque boîte utilisée par le programme a un nom et contient une valeur.

État de l'exécution d'un programme

On appelle *état de l'exécution d'un programme* le triplet formé par le nombre de boîtes utilisées, le nom de chacune d'elles et la valeur qu'elle contient.

Exécuter une instruction a pour effet de transformer cet état.

- Exécuter l'affectation `v = e` a pour effet de calculer la valeur de l'expression `e` dans l'état courant et de modifier cet état en mettant cette valeur dans la boîte de nom `v`. Par exemple, exécuter l'instruction `x = 4` dans l'état 1 produit l'état 2 (voir ci-dessous). De même, exécuter l'instruction `x = y + 3` dans l'état 1 produit l'état 3.



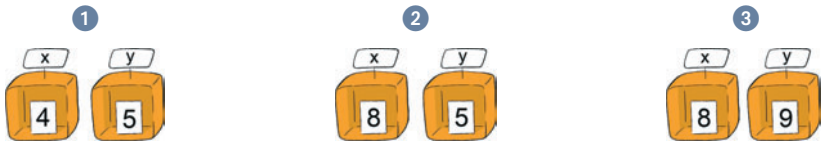
- Exécuter l'instruction d'entrée `v = int(input())` où `v` est une variable a pour effet d'interrompre le déroulement du programme jusqu'à ce que l'utilisateur tape un nombre entier au clavier. Ce nombre est alors mis dans la boîte de nom `v`. Des instructions similaires, `v = float(input())` et `v = input()` permettent à l'utilisateur de taper un nombre à virgule ou une chaîne de caractères.
- Exécuter l'instruction de sortie `print(e,end="")` où `e` est une expression ne modifie pas l'état, mais a pour effet de calculer la valeur de l'expression `e` dans l'état courant et d'afficher cette valeur à l'écran. Exécuter l'instruction de sortie `print(e)`, sans le `end=""`, affiche la valeur de l'expression `e` puis passe à la ligne. Exécuter l'instruction de sortie `print()` n'affiche rien mais passe à la ligne.
- Exécuter la séquence

```
p
q
```

où `p` et `q` sont deux instructions a pour effet d'exécuter `p` puis `q` dans l'état obtenu. Par exemple, exécuter l'instruction :

```
x = 8
y = 9
```

dans l'état 1 exécute l'instruction `x = 8` ce qui produit l'état 2 puis l'instruction `y = 9` ce qui produit l'état 3.



COMPRENDRE Une instruction composée mais unique

Attention, l'instruction :

```
x = 8
y = 9
```

est une unique instruction, à savoir une séquence de deux instructions plus petites :

```
x = 8
```

et

```
y = 9
```

- Exécuter le test :

```
if e:
  p
else:
  q
```

où `e` est une expression et `p` et `q` sont deux instructions a pour effet de calculer la valeur de l'expression `e`, puis d'exécuter l'instruction `p` ou l'instruction `q`, selon que la valeur de `e` est `True` (vrai) ou `False` (faux). Par exemple, exécuter l'instruction :

```
if x < 7:
  print("un peu")
else:
  print("beaucoup")
```

dans l'état 1 affiche `un peu`, car la valeur de l'expression `x < 7` dans cet état est `True`. En revanche, exécuter cette instruction dans l'état 2 affiche `beaucoup`.

Une variante du test est le test *sans* `else` :

```
if e:
  p
```

où `e` est une expression et `p` est une instruction. Exécuter cette instruction a pour effet de calculer la valeur de l'expression `e`, puis d'exécuter l'instruction `p` si la valeur de `e` est `True`. Par exemple, exécuter l'instruction :

```
if x < 7:
  print("un peu")
```

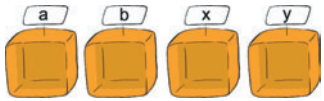
dans l'état 1 affiche `un peu`, alors qu'exécuter cette instruction dans l'état 2 n'affiche rien. Les expressions `p` et `q` doivent être à la ligne et deux colonnes à droite des mots `if` et `else`.

- Exécuter un programme complet `p` a pour effet de construire un état qui contient une boîte pour chaque variable affectée dans le programme `p`, puis d'exécuter l'instruction `p` dans cet état. Par exemple, exécuter le programme :

```
a = 4
b = 7
print("À vous de jouer")
x = int(input())
y = int(input())
if x == a and y == b:
  print("Coulé")
else:
  if x == a or y == b:
    print("En vue")
  else:
    print("À l'eau")
```

a pour effet de créer l'état

1



puis d'exécuter dans cet état l'instruction

```
a = 4
b = 7
print("À vous de jouer")
x = int(input())
y = int(input())
if x == a and y == b:
  print("Coulé")
else:
  if x == a or y == b:
    print("En vue")
  else:
    print("À l'eau")
```

SAVOIR-FAIRE Initialiser les variables

Identifier la première ligne du programme où une variable est utilisée. Lorsque l'algorithme comporte des tests, il peut y avoir plusieurs telles lignes pour la même variable, en fonction du résultat du test. Vérifier que cette première ligne est toujours une initialisation, c'est-à-dire qu'elle donne une valeur à la boîte de la variable. Une initialisation peut être rendue difficile à repérer parce qu'elle demande une saisie au clavier. Enfin, si une initialisation est manquante, il faut déterminer une valeur d'initialisation cohérente avec la suite du programme et ajouter l'instruction correspondante avant la première utilisation de la variable.

Exercice 1.6 (avec corrigé)

Quel problème peut se poser avec le programme suivant ? Le corriger.

```
f = f * 1
f = f * 2
f = f * 3
f = f * 4
f = f * 5
print(f)
```

La variable `f`, qui sert à calculer la factorielle de `5`, n'est pas initialisée et le résultat sera donc faussé par la valeur qu'elle contient par défaut. Les opérations qui sont effectuées sur `f` sont toutes des multiplications. Pour que la valeur initiale de `f` n'ait pas d'influence sur ces calculs, il faut que ce soit 1. On ajoutera donc l'instruction `f = 1` au début du programme. Cette initialisation est d'ailleurs cohérente avec la convention selon laquelle 0 ! = 1.

SAVOIR-FAIRE Comprendre un programme et expliquer ce qu'il fait

Identifier le rôle de chacune des variables utilisées. Si nécessaire, dérouler à la main une exécution du programme en notant l'état de l'exécution du programme au fur et à mesure.

Exercice 1.7 (avec corrigé)

Que fait ce programme ?

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
if b == 0 or d == 0:
  print("Dénominateur nul interdit !")
else:
  print(a * c)
  print(b * d)
```

Il y a ici quatre entrées `a`, `b`, `c` *et* `d`, *et deux sorties qui correspondent aux produits* `a * c` *et* `b * d`. *Le premier test indique que ni* `b` *ni* `d` *ne doivent être nulles. De tous ces éléments, on déduit que les entrées représentent sans doute les fractions* `a / b` *et* `c / d`, *que le programme calcule le produit de ces deux fractions, lorsqu'elles existent, et donne à nouveau le résultat sous la forme d'une fraction. On notera que ce qui peut rendre ce programme difficile à lire est, entre autres choses, les noms peu parlants choisis pour les variables. On gagnerait ainsi à renommer* `a` *en* `numerateur1`, `b` *en* `denominateur1`, `c` *en* `numerateur2`, *et* `d` *en* `denominateur2`.

Exercice 1.8

Que fait ce programme ? Comment devrait-on renommer ses variables ?

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
if b == 0 or d == 0:
  print("Dénominateur nul interdit !")
else:
  print(a * d + c * b)
  print(b * d)
```

Exercice 1.9

L'exécution de l'instruction :

```
x = 4
y = x + 1
x = 10
print(y)
```

produit-elle l'affichage de la valeur 5 ou de la valeur 11 ?

SAVOIR-FAIRE Écrire un programme

Identifier les entrées et les sorties du programme et les variables intermédiaires dont on aura besoin. Si le programme doit « prendre une décision », une ou plusieurs instructions de tests sont nécessaires.

Exercice 1.10 (avec corrigé)

Écrire un programme qui, étant donné une équation du second degré, détermine le nombre de ses solutions réelles et leurs valeurs éventuelles.

L'entrée est une équation du second degré `a` x^2 `+ b x + c = 0`, *fournie sous la forme de ses coefficients* `a`, `b` *et* `c`. *La sortie sera l'affichage du nombre de solutions réelles et de leurs valeurs. Le rôle du discriminant* Δ `= b`^{2} `- 4ac` *est ici suffisamment important pour mériter une variable intermédiaire* `delta` *qui stocke sa valeur.*

Il faut distinguer trois cas selon le signe du discriminant, ce qui se fait bien entendu à l'aide de tests.

```
from math import *

a = float(input())
b = float(input())
c = float(input())
delta = b * b - 4 * a * c
if delta < 0.0:
  print("Pas de solution")
elif delta == 0.0:
  print("Une solution : ",end="")
  print(- b / (2 * a))
else:
  print("Deux solutions : ",end="")
  print((- b - sqrt(delta)) / (2 * a),end="")
  print(" et ",end="")
  print((- b + sqrt(delta)) / (2 * a))
```

On reviendra, page 26, sur la première ligne du programme `from math import *`.

Exercice 1.11

Essayer le programme ci-dessus avec les entrées `a` = 1.0, `b` = 0.0, `c` = 1.0E-10 et `a` = 1.0, `b` = 0.0, `c` = -1.0E-10. Montrer qu'une infime variation sur l'un des coefficients permet de franchir la ligne qui sépare les cas où l'équation a des solutions des cas où elle n'en a pas.

Essayer le programme ci-dessus avec les entrées `a` = 1.0, `b` = 6.0, `c` = 9.0 et `a` = 0.1, `b` = 0.6, `c` = 0.9. Expliquer les résultats.

SAVOIR-FAIRE Mettre un programme au point en le testant

Pour vérifier si un programme ne produit pas d'erreur au cours de son exécution et s'il effectue réellement la tâche que l'on attend de lui, une première méthode consiste à exécuter plusieurs fois ce programme, en lui fournissant des entrées, appelées *tests*, qui permettent de détecter les erreurs éventuelles. Pour qu'elles jouent leur rôle, il faut choisir ces entrées de sorte que :

- on sache quelle doit être la sortie correcte du programme avec chacune de ces entrées,
- chaque cas distinct d'exécution du programme soit parcouru avec au moins un choix d'entrées,
- les cas limites soient essayés : par exemple le nombre 0, la chaîne vide ou à un seul caractère.

Exercice 1.12 (avec corrigé)

Proposer un jeu de tests satisfaisant pour le programme de la bataille navale.

Au minimum, il faut vérifier que le bateau est effectivement coulé si l'on donne les bonnes coordonnées, mais non coulé si l'on en donne de mauvaises. Par ailleurs, il faut tester si le programme affiche correctement `En vue`*, et donc tester au moins une case dans la même colonne que le bateau et une case dans la même ligne. Ces deux derniers tests permettront également de vérifier que les instructions conditionnelles du programme sont écrites correctement, et que par exemple il ne suffit pas d'avoir trouvé la bonne ligne pour couler le bateau. On testera donc le programme sur les entrées suivantes, par exemple, avec les résultats attendus :*

- *(4 ; 7) :* `Coulé`*,*
- *(1 ; 2) :* `À l'eau`*,*
- *(4 ; 9) :* `En vue` *(même ligne),*
- *(8 ; 7) :* `En vue` *(même colonne).*

On pourrait également tester ce qu'il se passe si l'on entre une coordonnée décimale ou une coordonnée qui dépasse les limites du tableau de jeu.

Exercice 1.13

Proposer un jeu de tests satisfaisant pour le programme de calcul des solutions réelles d'une équation du second degré ci-avant.

Les instructions et les expressions

L'affectation `x = y + 3` est une instruction. Elle est composée d'une variable `x` et d'une *expression* `y + 3`.

On attribue une *valeur* à chaque expression. Pour les expressions sans variables, comme `(2 + 5) * 3`, dont la valeur est 21, la valeur s'obtient simplement en effectuant les opérations présentes dans l'expression, dans cet exemple une addition et une multiplication. La valeur d'une expression qui contient des variables, par exemple `(2 + x) * 3`, se définit de la même manière, mais dépend de l'état dans lequel on calcule cette valeur.

Par exemple, la valeur de l'expression `(2 + x) * 3` dans l'état 1 est 15, alors que celle de cette même expression dans l'état 2 est 18.

Exercice 1.14

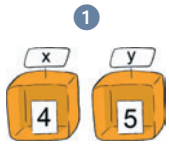
Les suites de symboles suivantes sont-elles des instructions ou des expressions ?

- `x`
- `x = y`
- `x = y + 3`
- `x + 3`
- `print(x + 3)`
- `x = int(input())`
- `x == a`
- `x == a and y == b`

Exercice 1.15

Déterminer la valeur des expressions suivantes dans l'état 1.

- `y + 3`
- `x + 3`
- `x + y`
- `x * x`
- `y == 5`
- `x == 3 and y == 5`



Exercice 1.16

Déterminer dans chacun des cas suivants tous les états tels que :

- `y - 2` vaut 1,
- `x * x` vaut 4,
- `x + y` vaut 1,
- ces trois conditions à la fois.

Les opérations

Les expressions sont formées en utilisant les opérations suivantes :

`+`	Addition entière
`-`	Soustraction entière
`*`	Multiplication entière
`//`	Quotient de la division euclidienne
`%`	Reste de la division euclidienne
`+`	Addition décimale

`-`	Soustraction décimale
`*`	Multiplication décimale
`/`	Division décimale
`pow`	Puissance
`sqrt`	Racine
`pi`	π
`sin`	Sinus
`cos`	Cosinus
`exp`	Exponentielle
`log`	Logarithme népérien
`abs`	Valeur absolue
`min`	Minimum
`max`	Maximum
`floor`	Partie entière
`random`	Nombre aléatoire décimal entre 0 et 1, selon la loi uniforme
`==`	Égal
`!=`	Différent
`<=`	Inférieur ou égal
`<`	Inférieur strictement
`>=`	Supérieur ou égal
`>`	Supérieur strictement
`len`	Longueur d'une chaîne de caractères
`s[n]`	`n`-ième élément de la chaine de caractères `s`
`chr`	Prend en argument un entier `n` et retourne une chaîne de caractères qui contient un unique caractère dont le code ASCII (voir le chapitre 8) est `n`.
`ord`	Fonction inverse de la précédente, qui prend en argument une chaîne de caractères `s` constituée d'un seul caractère et retourne le code ASCII du premier caractère de cette chaîne.
`+`	Concaténation. S'applique à deux chaînes de caractères et construit une unique chaîne formée de la première, suivie de la seconde.
`not`	Non
`and`	Et (variante `&`)
`or`	Ou (variante `|`)

En pratique

Les opérations `pow`, `sqrt`, `pi`, `sin`, `cos`, `exp`, `log` et `floor` ne font pas partie du langage Python mais d'une de ses extensions appelée `math`. Il faut donc ajouter au début du programme la commande `from math import *`, si on veut les utiliser. De même, utiliser l'opération `random` demande d'ajouter la commande `from random import *`.

Aller plus loin **Les opérations & et |**

L'opération `&` est une variante de l'opération `and`. La valeur de l'expression `t and u` est `False` quand la valeur de `t` est `False`, même si la valeur de l'expression `u` n'est pas définie, alors que la valeur de l'expression `t & u` n'est pas définie quand celle de `t` ou celle de `u` n'est pas définie. L'utilisation du `&` permet d'éviter de calculer la valeur de l'expression `u` si celle de `t` est `False`.
Ainsi l'exécution de l'instruction :

```
print(x !=0 & 1/x > 2)
```

provoque une erreur, quand `x` est égal à `0`, mais celle de l'instruction :

```
print(x != 0 and 1/x > 2)
```

affiche `False`.

De même, l'opération `|` est une variante de l'opération `or`. La valeur de l'expression `t or u` est `True` quand la valeur de `t` est `True`, même si la valeur de l'expression `u` n'est pas définie, alors que la valeur de `t | u` n'est pas définie quand celle de `t` ou celle de `u` n'est pas définie.

Exercice 1.17

Le but de cet exercice est d'écrire un programme qui demande à l'utilisateur une date comprise entre le 1er janvier 1901 et le 31 décembre 2099 et qui indique le nombre de jours écoulés entre le premier janvier 1901 et cette date.

Une bonne approximation de ce nombre est `(a - 1901) * 365 + (m - 1) * 30 + j - 1`. Mais il faut lui ajouter deux termes correctifs. Le premier est dû au fait que tous les mois ne font pas trente jours. On peut montrer que ce terme correctif vaut `m // 2` quand `m` est compris entre 1 et 2 et `(m + m // 8) // 2 - 2` quand `m` est compris entre 3 et 12. Le second est dû aux années bissextiles. On peut montrer que ce terme correctif vaut `(a - 1900)//4 - 1` si `a` est un multiple de 4 et `m` est compris entre 1 et 2 et vaut `(a - 1900)//4` sinon.

- Écrire un programme qui demande à l'utilisateur trois nombres qui constituent une date comprise entre le premier janvier 1901 et le 31 décembre 2099, par exemple 20 / 12 / 1996, et qui indique le nombre de jours écoulés entre le premier janvier 1901 et cette date.
- Écrire un programme qui demande à l'utilisateur deux dates et indique le nombre de jours écoulés entre ces deux dates.
- Sachant que le premier janvier 1901 était un mardi, écrire un programme qui demande à l'utilisateur une date et indique le jour de la semaine correspondant à cette date.

Exercice 1.18

En utilisant la fonction `random`, écrire un programme qui simule la loi uniforme sur l'intervalle [*a* ; *b*], où *a* et *b* sont deux réels donnés.

Exercice 1.19

En utilisant la fonction `random`, écrire un programme qui affiche aléatoirement `pile` ou `face` de façon équiprobable.

ALLER PLUS LOIN **Les ordinateurs et le hasard**

Parmi les opérations de base, on a cité la fonction `random`, qui renvoie un nombre aléatoire compris entre 0 et 1. Si l'on s'y arrête quelques secondes, l'existence d'une telle fonction est contradictoire avec la notion même d'algorithme : un processus suffisamment bien décrit et détaillé pour être exécuté sans erreur ni initiative de la part d'une machine ne peut pas mener à un résultat imprévisible et différent à chaque exécution. Pourtant l'introduction de hasard dans les programmes est indispensable, par exemple pour créer des situations imprévues dans les logiciels de jeux, mais aussi pour résoudre certains problèmes qui ne peuvent pas être résolus sans une part de hasard, comme on le verra au chapitre 16. La fonction `random` ne génère pas de nombres réellement aléatoires, mais les résultats obtenus sont suffisamment proches de tirages aléatoires pour la plupart des applications qui utilisent ce genre de nombres. L'exercice 2.8 donne un exemple d'un tel générateur de nombres pseudo-aléatoires.

L'indentation

L'expression `x - y + z` peut être construite ou bien avec le signe `+` et les deux expressions `x - y` et `z`, ou alors avec le signe `-` et les deux expressions `x` et `y + z`. Comme en mathématiques, on lève cette ambiguïté en utilisant des parenthèses : on écrit la première expression `(x - y) + z` et la seconde `x - (y + z)`. Et, comme en mathématiques, on décide que, quand on n'utilise pas de parenthèses, l'expression `x - y + z` signifie `(x - y) + z` et non `x - (y + z)`.

Un problème similaire se pose avec les instructions, et l'ambiguïté est levée en Python non par l'usage de parenthèses mais grâce à l'indentation, c'est-à-dire grâce au décalage de certaines lignes vers la droite par l'utilisation d'espaces en début de ligne. Ainsi, l'instruction

```
if e:
  p
else:
  q
  r
```

n'a pas le même sens que l'instruction :

```
if e:
  p
else:
  q
r
```

Dans le premier cas l'instruction `r` n'est exécutée que si la valeur de l'expression `e` est `False` (faux) ; dans le second cas elle est toujours exécutée. L'indentation du programme joue le rôle des parenthèses pour les expressions.

En pratique L'indentation selon les langages

En Python, l'indentation est partie intégrante de la syntaxe. Dans la plupart des autres langages de programmation, ce sont des accolades qui sont utilisées pour lever les ambiguïtés, l'indentation étant alors facultative.

Exercice 1.20

Quel est le résultat de l'exécution des instructions :

```
if x == 4:
  y = 1
else:
  y = 2
  z = 3
```

et

```
if x == 4:
  y = 1
else:
  y = 2
z = 3
```

dans l'état :

Exercice 1.21

Trouver deux programmes qui ne diffèrent que par l'indentation et dont l'exécution est différente.

Exercice 1.22 (avec corrigé)

Écrire un programme qui affiche le tarif du timbre à poser sur une lettre en fonction de son type et de son poids.

On trouve sur le site web de la Poste le tableau suivant (au 1er octobre 2011) :

Poids jusqu'à	Lettre verte	Lettre prioritaire	Ecopli
20 g	0,57 €	0,60 €	0,55 €
50 g	0,95 €	1,00 €	0,78 €
100 g	1,40 €	1,45 €	1,00 €

On peut par exemple considérer que les différentes gammes de poids sont des sous-cas dans chaque type de lettre. Le programme est donc constitué de trois tests correspondant aux différents types de lettres, l'instruction exécutée dans chacun des cas étant elle-même constituée de trois tests correspondant aux différents poids. On décide de ne rien afficher quand les entrées `type` *et* `poids` *ne correspondent pas à une catégorie du tableau.*

```
type = input()
poids = int(input())
if type == "verte":
   if poids <= 20:
      print(0.57)
   elif poids <= 50:
      print(0.95)
   elif poids <= 100:
      print(1.40)
elif type == "prioritaire":
   if poids <= 20:
      print(0.60)
   elif poids <= 50:
      print(1.00)
   elif poids <= 100:
      print(1.45)
elif type == "ecopli":
   if poids <= 20:
      print(0.55)
   elif poids <= 50:
      print(0.78)
   elif poids <= 100:
      print(1.00)
```

Ai-je bien compris ?

- Quelles sont les trois instructions présentées dans ce chapitre et qui permettent de construire un programme élémentaire ?
- Quelle est la différence entre une instruction et une expression ?
- Comment l'exécution d'une instruction transforme-t-elle l'état de l'exécution du programme ?

7

Le livre de l'addition et de la soustraction d'après le calcul indien de **Muhammad al-Khwarizmi** (783 ?-850 ?), qui présente la numération décimale à position et des algorithmes permettant d'effectuer les opérations sur les nombres exprimés dans ce système, a été le vecteur de la diffusion de ce système de numération dans le bassin méditerranéen. Le mot algorithme est dérivé du nom d'al-Khwarizmi et le mot algèbre (*al-jabr*) du titre d'un autre de ses livres.

Représenter des nombres entiers et à virgule

Au commencement étaient le 0 et le 1, puis nous créâmes les nombres, les textes, les images et les sons.

Dans ce chapitre, nous voyons comment les nombres sont représentés dans les ordinateurs avec des 0 et des 1.

Nous introduisons la notion de base, en partant de la notation décimale que nous utilisons ordinairement pour écrire les nombres entiers. Nous passons par la base cinq puis décrivons la base deux, aussi appelée représentation binaire. Nous généralisons ensuite aux nombres relatifs en utilisant la notation en complément à deux, puis aux nombres à virgule, représentés par leur signe, leur mantisse et leur exposant.

Vus de l'extérieur, les ordinateurs et les programmes que l'on utilise tous les jours permettent de mémoriser, de transmettre et de transformer des nombres, des textes, des images, des sons, etc.

Pourtant, quand on les observe à une plus petite échelle, ces ordinateurs ne manipulent que des objets beaucoup plus simples : des 0 et des 1. Mémoriser, transmettre et transformer des nombres, des textes, des images ou des sons demande donc d'abord de les représenter comme des suites de 0 et de 1.

La mémoire des ordinateurs est constituée d'une multitude de petits circuits électroniques qui ne peuvent être, chacun, que dans deux états (voir le chapitre 14). Comme il fallait donner un nom à ces états, on a décidé de les appeler 0 et 1, mais on aurait pu tout aussi bien les appeler *A* et *B*, froid et chaud ou faux et vrai. Une telle valeur, 0 ou 1, s'appelle un *booléen*, un *chiffre binaire* ou encore un *bit* (*binary digit*). Un tel circuit à deux états s'appelle un *circuit mémoire un bit* et son état se décrit donc par le symbole 0 ou par le symbole 1.

L'état d'un circuit, composé de plusieurs de ces circuits mémoire un bit, se décrit par une suite finie de 0 et de 1, que l'on appelle un *mot*. Par exemple, le mot 100 décrit l'état d'un circuit composé de trois circuits mémoire un bit, respectivement dans l'état 1, 0 et 0.

Exercice 7.1

On imagine un ordinateur dont la mémoire est constituée de quatre circuits mémoire un bit. Quel est le nombre d'états possibles de la mémoire de cet ordinateur ? Même question pour un ordinateur dont la mémoire est constituée de dix circuits mémoire un bit. Et pour un ordinateur dont la mémoire est constituée de 34 milliards de tels circuits.

Exercice 7.2

On veut représenter chacune des sept couleurs de l'arc-en-ciel par un mot, les sept mots devant être distincts et de même longueur. Quelle est la longueur minimale de ces mots ?

Exercice 7.3

Trouvez trois informations de la vie courante qui peuvent être exprimées par un booléen.

Exercice 7.4

On considère une « box internet » avec une diode électroluminescente, éteinte ou allumée selon un motif de 0 et de 1 qui change à chaque demi-seconde. Lorsque la box est éteinte, la diode aussi, le motif est 0000000000… Lorsque la box est allumée et fonctionne, la diode est allumée en continu, le motif vaut donc 1111111111… Lorsque la box est en panne, le fournisseur d'accès souhaite que la diode clignote selon différents motifs en fonction du type de panne : pas de réseau, réseau saturé, facture non payée, etc. On parle ainsi de clignotement rapide pour le motif 0101010101.

1. Proposer deux motifs qui pourraient correspondre aux descriptions « clignotement lent » et « clignotement très lent ».

2. Comment peut-on décrire les motifs suivants, répétés indéfiniment : 0000100000 et 0101010000 ?
3. Comment faut-il procéder pour qu'un motif donné ne s'affiche que toutes les dix secondes ?
4. Dans une situation où il y aurait deux types de panne en même temps, comment pourrait-on procéder pour afficher les motifs correspondant aux deux messages d'erreurs différents ?

La représentation des entiers naturels

Depuis le Moyen Âge, on écrit les nombres entiers naturels en *notation décimale à position*. Cela signifie que, pour écrire le nombre entier naturel n, on commence par imaginer n objets, que l'on groupe par paquets de dix, puis on groupe ces paquets de dix objets en paquets de dix paquets, etc. À la fin, il reste entre zéro et neuf objets isolés, entre zéro et neuf paquets isolés de dix objets, entre zéro et neuf paquets isolés de cent, etc. Et on écrit cet entier naturel en notant, de droite à gauche, le nombre d'objets isolés, le nombre de paquets de dix, le nombre de paquets de cent, le nombre de paquets de mille, etc. Chacun de ces nombres étant compris entre zéro et neuf, seuls dix chiffres sont nécessaires : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. Par exemple, l'écriture 2359 exprime un entier naturel formé de 9 unités, 5 dizaines, 3 centaines et 2 milliers.

Le choix de faire des paquets de dix est peut-être dû au fait que l'on a dix doigts, mais on aurait pu tout aussi bien décider de faire des paquets de deux, de cinq, de douze, de vingt, de soixante, etc. On écrirait alors les nombres entiers naturels en notation à position en base deux, cinq, douze, vingt ou soixante. La notation décimale à position s'appelle donc aussi la notation à position en base dix.

En *notation binaire*, c'est-à-dire en notation à position en base deux, le nombre treize s'écrit 1101 : de droite à gauche, 1 unité, 0 deuzaine, 1 quatraine et 1 huitaine. L'écriture d'un entier naturel en binaire est en moyenne 3,2 fois plus longue que son écriture en base dix, mais elle ne demande d'utiliser que deux chiffres : 0 et 1.

Indiquer la base

Dans ce livre, quand une suite de chiffres exprime un nombre dans une base différente de dix, on indique la base entre parenthèses, par exemple : 1101 (en base deux). On souligne aussi parfois un mot pour indiquer qu'il exprime un nombre en base deux : 1101. Enfin, on rassemble parfois les bits par groupe de quatre ou de huit dans les mots très longs pour qu'ils soient plus faciles à lire : 1111111101 est écrit 11 1111 1101. Comme en base dix, ces groupes sont formés de droite à gauche.

Exercice 7.5

Un horloger excentrique a eu l'idée de fabriquer une montre sur laquelle l'heure est indiquée par 10 diodes électroluminescentes appelées 1 h, 2 h, 4 h, 8 h, 1 min, 2 min, 4 min, 8 min, 16 min et 32 min. Pour connaître l'heure, il suffit d'ajouter la valeur de toutes les diodes allumées.

Quelle heure est-il quand sont allumées les diodes 1 h, 2 h, 4 h, 1 min, 2 min, 8 min, 16 min et 32 min ? Quelles sont les diodes allumées à 5 h 55 ? Est-il possible de représenter toutes les heures ? Toutes les configurations sont-elles la représentation d'une heure ?

Comme la mémoire des ordinateurs est constituée de circuits qui ne peuvent être chacun que dans deux états, on peut utiliser chaque circuit de la mémoire pour représenter un chiffre binaire, en identifiant l'un de ces états avec le chiffre binaire 0 et l'autre avec le chiffre binaire 1 – on comprend *a posteriori* pourquoi on a choisi d'appeler ces états eux-mêmes 0 et 1. Le nombre 13 = $\underline{1101}$ est donc représenté dans la mémoire d'un ordinateur par le mot 1101, c'est-à-dire par quatre circuits respectivement dans les états 1, 0, 1 et 1.

La base cinq

Pour comprendre comment transformer un entier naturel, écrit en base dix, dans une autre base, on commence par la base cinq, moins particulière que la base deux, sur laquelle on reviendra plus tard.

SAVOIR-FAIRE Trouver la représentation en base cinq d'un entier naturel donné en base dix

Pour écrire les entiers naturels en base cinq, on a besoin de cinq chiffres : 0, 1, 2, 3, 4. Quand on a n objets, on les groupe par paquets de cinq, qu'on groupe ensuite en paquets de cinq paquets, etc. Autrement dit, on fait une succession de divisions par 5, jusqu'à obtenir un quotient égal à 0.

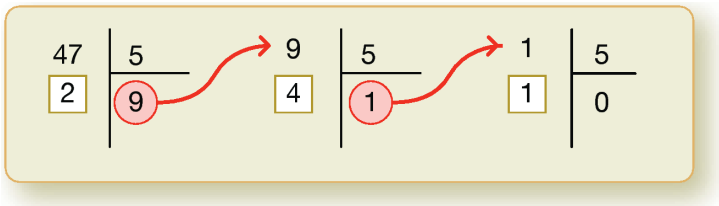
Exercice 7.6 (avec corrigé)

Trouver la représentation en base cinq de 47.

47 objets se regroupent en 9 paquets et 2 unités, puis les 9 paquets se regroupent en 1 paquet de paquets et 4 paquets.

$47 = 9 \times 5 + 2 = (1 \times 5 + 4) \times 5 + 2 = (1 \times 5^2) + (4 \times 5^1) + (2 \times 5^0)$

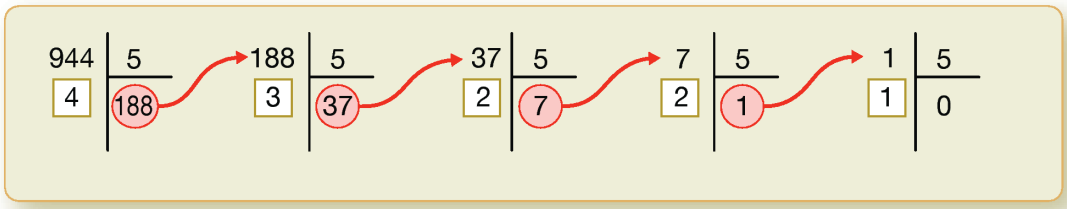
Donc 47 = 142 (en base cinq).



Exercice 7.7 (avec corrigé)

Trouver la représentation en base cinq du nombre 944.

On obtient 944 = 12234 (en base cinq).



Exercice 7.8

Trouver la représentation en base cinq du nombre 289.

SAVOIR-FAIRE Trouver la représentation en base dix d'un entier naturel donné en base cinq

Pour trouver la représentation en base dix d'un entier naturel donné en base cinq, on utilise le fait qu'en base cinq, le chiffre le plus à droite représente les unités, le précédent les paquets de 5, le précédent les paquets de $5 \times 5 = 5^2 = 25$, le précédent de $5 \times 5 \times 5 = 5^3 = 125$, et ainsi de suite.

Exercice 7.9 (avec corrigé)

Trouver la représentation en base dix du nombre 401302 (en base cinq).

401302 (en base cinq) = $(\mathbf{2} \times 5^0) + (\mathbf{0} \times 5^1) + (\mathbf{3} \times 5^2) + (\mathbf{1} \times 5^3) + (\mathbf{0} \times 5^4) + (\mathbf{4} \times 5^5) = 12702$.

Exercice 7.10

Trouver la représentation en base dix des nombres 2341 (en base cinq) et 444 (en base cinq).

La base deux

Les nombres exprimés en base deux sont plus difficiles à lire, car il n'y a que deux chiffres, 0 et 1, mais le principe de la numération en base deux est en tout point similaire à celui de la numération en base cinq.

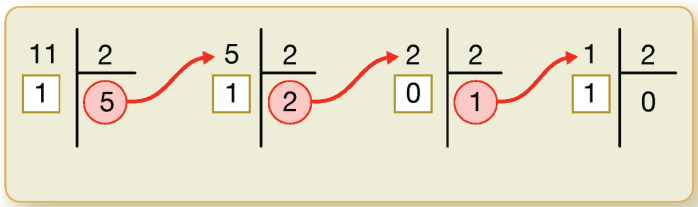
SAVOIR-FAIRE Trouver la représentation en base deux d'un entier naturel donné en base dix

Pour écrire les nombres en base deux, on a besoin de deux chiffres : 0 et 1. Quand on a n objets, on les groupe par paquets de deux, qu'on regroupe eux-mêmes en paquets de deux paquets, etc. Autrement dit, on fait une succession de divisions par 2, jusqu'à obtenir un quotient égal à 0.

Exercice 7.11 (avec corrigé)

Trouver la représentation en base deux du nombre 11.

On obtient 11 = 1011.



Exercice 7.12

Trouver la représentation en base deux du nombre 1000.

Exercice 7.13

Chercher sur le Web la date de la mort de Charlemagne. Trouver la représentation en base deux de ce nombre.

Exercice 7.14

Donner les représentations en base deux des nombres 1, 3, 7, 15, 31 et 63. Expliquer le résultat.

SAVOIR-FAIRE Trouver la représentation en base dix d'un entier naturel donné en base deux

Pour trouver la représentation en base dix d'un entier naturel donné en base deux, on utilise le fait qu'en base deux, le chiffre le plus à droite représente les unités, le précédent les paquets de 2, le précédent les paquets de $2 \times 2 = 2^2 = 4$, le précédent de $2 \times 2 \times 2 = 2^3 = 8$, etc.

Exercice 7.15 (avec corrigé)

Trouver la représentation en base dix du nombre 11111111.

11111111 = $(\mathbf{1} \times 2^0) + (\mathbf{1} \times 2^1) + (\mathbf{1} \times 2^2) + (\mathbf{1} \times 2^3) + (\mathbf{1} \times 2^4) + (\mathbf{1} \times 2^5) + (\mathbf{1} \times 2^6) + (\mathbf{1} \times 2^7)$ = *255.*

Exercice 7.16

Trouver la représentation en base dix du nombre 10010110.

Exercice 7.17

C'est en 11110010000 qu'a été démontré le théorème fondamental de l'informatique. Exprimer ce nombre en base dix.

Exercice 7.18

Montrer qu'avec un mot de n bits on peut représenter les nombres de 0 à 2^n–1.

Les octets

Dans la mémoire des ordinateurs les circuits mémoire un bit sont souvent groupés par huit : les *octets*. On utilise souvent des nombres exprimés en notation binaire, c'est-à-dire en base deux, sur un, deux, quatre ou huit octets, soit 8, 16, 32 ou 64 bits. Ceci permet de représenter les nombres de 0 à 1111 1111 = 255 sur un octet, de 0 à 1111 1111 1111 1111 = 65 535 sur deux octets, de 0 à 1111 1111 1111 1111 1111 1111 1111 1111 = 4 294 967 295 sur quatre octets et de 0 à 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 = 18 446 744 073 709 551 615 sur huit octets.

Exercice 7.19

Pour multiplier par dix un entier naturel exprimé en base dix, il suffit d'ajouter un 0 à sa droite, par exemple, 12 × 10 = 120. Quelle est l'opération équivalente pour les entiers naturels exprimés en base deux ? Exprimer en base deux les nombres 3, 6 et 12 pour illustrer cette remarque.

Exercice 7.20

Quelle est la représentation binaire du nombre 57 ? Et celle du nombre 198 ? Soit m un mot de 8 bits, n l'entier naturel représenté en binaire par le mot m, m' le mot obtenu en remplaçant dans m chaque 0 par un 1 et chaque 1 par un 0 et n' l'entier naturel représenté en binaire par le mot m'. Exprimer n et n' comme une somme de puissances de 2, montrer que

$n + n' = 255$. Montrer que la représentation binaire du nombre $255 - n$ est obtenue en remplaçant dans celle de n chaque 0 par un 1 et chaque 1 par un 0.

Une base quelconque

On peut généraliser à une base quelconque les méthodes vues pour la base cinq et la base deux.

SAVOIR-FAIRE Trouver la représentation en base k d'un entier naturel donné en base dix

Pour écrire les entiers naturels en base k, on a besoin de k chiffres. Quand on a n objets, on les groupe par paquets de k, qu'on regroupe à leur tour en paquets de k paquets, etc. Autrement dit, on fait une succession de divisions par k, jusqu'à obtenir un quotient égal à 0.

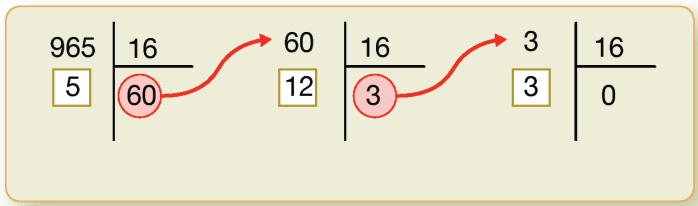
Base 16

En base seize, on a besoin de 16 chiffres : 0, 1, 2, 3, 4, 5 , 6, 7, 8, 9, puis a (dix), b (onze), c (douze), d (treize), e (quatorze) et f (quinze).

Exercice 7.21 (avec corrigé)

Trouver la représentation en base seize du nombre 965.

Réponse : 965 = 3c5 (en base seize)



Exercice 7.22

Trouver la représentation en base seize des nombres 6725 et 18 379.

SAVOIR-FAIRE Trouver la représentation en base dix d'un entier naturel donné en base *k*

Pour trouver la représentation en base dix d'un entier naturel donné en base k, on utilise le fait qu'en base k, le chiffre le plus à droite représente les unités, le précédent les paquets de k, le précédent les paquets de $k \times k = k^2$, le précédent les paquets de $k \times k \times k = k^3$, etc.

Exercice 7.23 (avec corrigé)

Trouver la représentation en base dix du nombre 4e2c (en base seize).

4e2c (en base seize) = $(\mathbf{12} \times 16^0) + (\mathbf{2} \times 16^1) + (\mathbf{14} \times 16^2) + (\mathbf{4} \times 16^3) = 20\,012$.

Exercice 7.24

Trouver la représentation en base dix des nombres abcd (en base seize) et 281ef (en base seize).

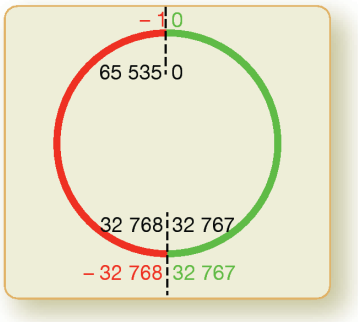
Exercice 7.25

Chercher sur le Web ce qu'est le système de numération Shadok. Est-ce un système de numération à position ? Si oui, en quelle base et avec quels chiffres ?

La représentation des entiers relatifs

Pour représenter les entiers relatifs en notation binaire, on doit étendre la représentation aux nombres négatifs. Une solution est de réserver un bit pour le signe de l'entier à représenter et d'utiliser les autres pour représenter sa valeur absolue. Ainsi, avec des mots de 16 bits, en utilisant 1 bit pour le signe et 15 bits pour la valeur absolue, on pourrait représenter les entiers relatifs de -111 1111 1111 1111 = $-(2^{15} - 1) = -32\,767$ à 111 1111 1111 1111 = $2^{15} - 1 = 32\,767$. Cependant, cette méthode a plusieurs inconvénients, l'un d'eux étant qu'il y a deux zéros, l'un positif et l'autre négatif.

On a donc préféré une autre méthode, qui consiste à représenter un entier relatif par un entier naturel. Si on utilise des mots de 16 bits, on peut représenter les entiers relatifs compris entre -32 768 et 32 767 : on représente un entier relatif x positif ou nul comme l'entier naturel x, et un entier relatif x strictement négatif comme l'entier naturel $x + 2^{16} = x + 65\,536$, qui est compris entre 32 768 et 65 535. Ainsi, les entiers naturels de 0 à 32 767 servent à représenter les entiers relatifs positifs ou nuls, en vert sur la figure, et les entiers naturels de 32 768 à 65 535 décrivent les entiers relatifs strictement négatifs, en rouge sur la figure.



L'entier relatif -1 est représenté comme l'entier naturel 65 535, c'est-à-dire par le mot 1111 1111 1111 1111.

Cette manière de représenter les entiers relatifs s'appelle la notation en *complément à deux*.

Avec des mots de seize bits, on écrit les entiers relatifs compris entre $-2^{15} = -32\,768$ et $2^{15} - 1 = 32\,767$. Plus généralement, avec des mots de n bits, on écrit les entiers relatifs compris entre -2^{n-1} et $2^{n-1} - 1$:

- un entier relatif x positif ou nul compris entre 0 et $2^{n-1} - 1$ est représenté par l'entier naturel x compris entre 0 et $2^{n-1} - 1$;
- un entier relatif x strictement négatif compris entre -2^{n-1} et -1 est représenté par l'entier naturel $x + 2^n$ compris entre 2^{n-1} et $2^n - 1$.

Exercice 7.26

Quels entiers relatifs peut-on représenter avec des mots de 8 bits ? Combien sont-ils ? Même question avec des mots de 32 bits et 64 bits.

SAVOIR-FAIRE Trouver la représentation binaire sur n bits d'un entier relatif donné en décimal

On a vu que :

- Si l'entier relatif x est positif ou nul, on le représente comme l'entier naturel x.
- S'il est strictement négatif, on le représente comme l'entier naturel $x + 2^n$.

Exercice 7.27 (avec corrigé)

Trouver la représentation binaire sur huit bits des entiers relatifs 0 et -128.

L'entier relatif 0 est représenté comme l'entier naturel 0 : 0000 0000.

L'entier relatif -128 est représenté comme l'entier naturel $-128 + 2^8 = -128 + 256 = 128$: 1000 0000.

Exercice 7.28

Trouver la représentation binaire sur huit bits des entiers relatif 127 et -127.

SAVOIR-FAIRE Trouver la représentation décimale d'un entier relatif donné en binaire sur n bits

Si cet entier relatif est donné par le mot m, on commence par calculer l'entier naturel p représenté par ce mot. Si p est strictement inférieur à 2^{n-1}, c'est l'entier relatif représenté, s'il est supérieur ou égal à 2^{n-1}, l'entier relatif représenté est $p - 2^n$.

Exercice 7.29 (avec corrigé)

Trouver la représentation décimale des entiers relatifs dont la représentation binaire sur huit bits est 0000 0000 et 1000 0000.

Le mot 0000 0000 représente l'entier naturel 0 et donc l'entier relatif 0. Le mot 1000 0000 représente l'entier naturel 128 = 2^7 et donc l'entier relatif 128 - 2^8 = 128 - 256 = -128.

Exercice 7.30

Trouver la représentation décimale des entiers relatifs dont la représentation binaire sur huit bits est 0111 1111 et 1000 0001.

SAVOIR-FAIRE Calculer la représentation p' de l'opposé d'un entier relatif x à partir de sa représentation p, pour une représentation des entiers relatifs sur huit bits

Si l'entier relatif x est compris entre 0 et 127, alors il est représenté sur huit bits par l'entier naturel $p = x$ et son opposé $-x$ est représenté par l'entier naturel $p' = -x + 2^8 = -x + 256 = 256 - p$.
Si l'entier relatif x est compris entre -127 et -1, alors il est représenté par l'entier naturel $p = x + 2^8 = x + 256$ et son opposé $-x$ est représenté par l'entier naturel $p' = -x = 256 - p$.
Donc, sauf quand x = -128, dont l'opposé n'est pas représentable sur 8 bits, si un entier relatif x est représenté par l'entier naturel p, son opposé $-x$ est représenté par l'entier naturel $p' = 256 - p = (255 - p) + 1$.
Calculer $255 - p$ = 1111 1111 - p est facile, puisqu'il suffit, dans la représentation binaire de p, de remplacer chaque 0 par un 1 et chaque 1 par un 0 (voir l'exercice 7.20). Il suffit ensuite d'ajouter 1 au nombre obtenu.

Exercice 7.31 (avec corrigé)

Calculer la représentation binaire sur huit bits de l'entier relatif 4, puis celle de son opposé.

L'entier relatif 4 est représenté comme l'entier naturel 4 = 0000 0100.

Pour calculer la représentation de son opposé, on remplace les 0 par des 1 et les 1 par des 0, ce qui donne 1111 1011. Puis on ajoute 1, ce qui donne 1111 1100. On peut vérifier que ce nombre est bien la représentation de l'entier relatif -4, c'est-à-dire de l'entier naturel -4 + 256 = 252.

Exercice 7.32

Calculer la représentation binaire sur huit bits de l'entier relatif -16, puis de son opposé.

Exercice 7.33

Montrer que le bit le plus à gauche vaut 1 pour les entiers relatifs strictement négatifs et 0 pour les entiers relatifs positifs ou nuls.

Exercice 7.34

On considère des entiers relatifs sur 3 bits. Dessiner le cercle rouge-vert ci-avant en plaçant les 8 nombres : 0, 1, 2, 3, -1, -2, -3, -4 à leur place. Relier les nombres opposés : 1 et -1, 2 et -2, etc. Quelle est l'interprétation géométrique de la fonction qui à chaque nombre associe son opposé ?

Exercice 7.35

Représenter les entiers relatifs 96 et 48 en binaire sur huit bits. Ajouter les deux nombres binaires obtenus en utilisant l'algorithme de l'addition binaire du chapitre 18. Quel est l'entier relatif obtenu ? Pourquoi est-il négatif ?

Exercice 7.36

Expliquer comment faire une soustraction de deux nombres binaires sur huit bits à partir du calcul de l'opposé et de l'algorithme de l'addition du chapitre 18. Calculer ainsi 15 - 7.

La représentation des nombres à virgule

Comme la notation décimale, la notation binaire permet aussi de représenter des nombres à virgule. En notation décimale, les chiffres à gauche de la virgule représentent des unités, des dizaines, des centaines, etc. et ceux à droite de la virgule, des dixièmes, des centièmes, des millièmes, etc. De même, en binaire, les chiffres à droite de la virgule représentent des demis, des quarts, des huitièmes, des seizièmes, etc. On peut ainsi représenter, par exemple, le nombre un et un quart : 1,01. Toutefois, cette manière de faire ne permet pas de représenter des nombres très grands ou très petits comme le nombre d'Avogadro ou la constante de Planck. On utilise donc une autre représentation similaire à la « notation scientifique » des calculatrices, sauf qu'elle est en base deux et non en base dix. Un nombre est représenté sous la forme $s\ m\ 2^n$ où s est le signe du nombre, n son exposant et m sa mantisse. Le signe est + ou -, l'exposant est un entier relatif et la mantisse est un nombre à virgule, compris entre 1 inclus et 2 exclu.

Par exemple, quand on utilise 64 bits pour représenter un nombre à virgule, on utilise 1 bit pour le signe, 11 bits pour l'exposant et 52 bits pour la mantisse. Le signe + est représenté par 0 et le signe - par 1. L'exposant n est un entier relatif compris entre -1 022 et 1 023 ; on le représente comme l'entier naturel n + 1 023, qui est compris entre 1 et 2 046. Les deux entiers naturels 0 et 2 047 sont réservés pour des situations exceptionnelles (+∞, -∞, NaN, etc.). La mantisse m est un nombre binaire à virgule compris entre 1 inclus et 2 exclu, comprenant 52 chiffres après la virgule. Comme cette mantisse est comprise entre 1 et 2, elle a toujours un seul chiffre avant la virgule et ce chiffre est toujours un 1 ; il est donc inutile de le représenter et on utilise les 52 bits pour représenter les 52 chiffres après la virgule.

ALLER PLUS LOIN **Valeurs exceptionnelles**

Ce codage prend en compte des valeurs exceptionnelles : +∞, -∞ et NaN (*not a number*) qui est une valeur indéfinie. Ces valeurs non numériques sont représentées respectivement par les mots de 64 bits suivants :

- 0 11111111111 0000000000000000000000000000000000000000000000000000
- 1 11111111111 0000000000000000000000000000000000000000000000000000
- 1 11111111111 1111111111111111111111111111111111111111111111111111

Ce codage permet également de représenter des valeurs infinitésimales qui, sans être nulles, sont trop petites pour que l'on puisse calculer avec elles.

SAVOIR-FAIRE Trouver la représentation en base dix d'un nombre à virgule donné en binaire

On identifie le signe, la mantisse et l'exposant.

Exercice 7.37 (avec corrigé)

Trouver le nombre à virgule représenté par le mot
1100010001101001001111000011100000000000000000000000000000000000

Le signe est représenté par 1.
L'exposant est représenté par 10001000110. La mantisse est représentée par 10010011110000111000000000 00000000000000000000000.
Le signe du nombre est donc -. Le nombre 100 0100 0110 est égal à 1 094 et l'exposant du nombre est n = *1094 - 1023 = 71. Sa mantisse est :*

$$\begin{aligned} m &= 1.1001\ 0011\ 1100\ 0011\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \\ &= 1 + 1/2 + 1/2^4 + 1/2^7 + 1/2^8 + 1/2^9 + 1/2^{10} + 1/2^{15} + 1/2^{16} + 1/2^{17} \\ &= (2^{17} + 2^{16} + 2^{13} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^2 + 2 + 1) / 2^{17} \\ &= 206727/131072. \end{aligned}$$

Le nombre représenté est donc -206 727 / 131 072 × 2^{71} = -3.724… × 10^{21}.

Exercice 7.38

Trouver le nombre à virgule représenté par le mot :
0001000000111101001110010101100000000000000000000000000000000000.

Exercice 7.39

Comment est représenté le nombre à virgule $2^{-1\,022}$ (qui est égal à $2{,}225\ldots \times 10^{-308}$) ?

Exercice 7.40

Comment est représenté le nombre entier 7 ? Et le nombre à virgule 7,0 ?

Exercice 7.41

À combien de décimales environ correspondent 52 chiffres binaires après la virgule ?

Exercice 7.42

Quel est le plus grand nombre à virgule que l'on peut représenter en binaire avec 64 bits ? Quel est le plus petit nombre à virgule, donc négatif, que l'on peut représenter en binaire avec 64 bits ? Quel est le plus petit nombre à virgule strictement positif que l'on peut représenter en binaire avec 64 bits ?

Exercice 7.43

Quelle précision perd-on si on divise par deux un nombre à virgule avant de le remultiplier par deux ?

Exercice 7.44

À chaque multiplication de deux nombres à virgule, on arrondit le calcul en ne gardant que 52 chiffres binaires après la virgule, ce qui introduit une erreur relative de l'ordre de 2^{-52}. Quelle est la valeur de cette erreur en base dix ? Si on fait plusieurs multiplications, ces erreurs s'accumulent. Quelle est l'erreur relative d'un calcul qui est formé d'un million de multiplications, qui dure quelques millisecondes sur un ordinateur usuel ?

Exercice 7.45

On considère le programme suivant :

```
x = 1.0
y = x + 1.0
while y - x == 1.0:
  x = x * 2.0
  y = x + 1.0
```

1. Si l'on calculait sur des nombres rationnels exacts, que se passerait-il lors de l'exécution de ce programme ?
2. Écrire ce programme et l'exécuter. Que constate-t-on ?
3. Modifier le programme de façon à déterminer au bout de combien d'exécutions du corps de la boucle il s'arrête, ainsi que la valeur de `x` à la fin de cette exécution.
4. Comment est représentée cette dernière valeur de `x` ? Et celle de `y` ?

5. Proposer une explication de ce comportement.

Exercice 7.46

On considère le programme suivant :

```
a = 0.0
for loop in range(0,10):
  a = a + 0.1
  print(a)
```

1. Si l'on calculait sur des nombres rationnels exacts, que se passerait-il lors de l'exécution de ce programme ?
2. Écrire ce programme et l'exécuter. Que constate-t-on ?
3. Vérifier que la représentation binaire de 0,1 est 00111111101110011001100110011001100110011001100110011001100110011010.
4. Quel nombre décimal cette représentation désigne-t-elle en réalité ?
5. En déduire les représentations binaires de 0,2, 0,3 et les nombres décimaux que cette représentation désigne en réalité.
6. Expliquer le résultat obtenu.

Ai-je bien compris ?

- En quelle base représente-t-on le plus souvent les nombres en informatique ? Pourquoi ?
- Comment représente-t-on les nombres négatifs ?
- Comment représente-t-on les nombres à virgule ?

8

Samuel Morse (1791-1872) est l'inventeur d'un code, dans lequel chaque lettre est exprimée par une alternance de sons brefs symbolisés par « . » et longs « – », utilisé pour télégraphier des textes. La lettre « a » y est exprimée par les sons « . – », la lettre « b » par les sons « – … », etc. Artiste peintre, Samuel Morse s'est intéressé aux télécommunications après qu'en 1925, un message lui annonçant que sa femme était malade ne lui est pas parvenu à temps. Comme nous le verrons au chapitre 12, le code morse est à références de longueurs variables, mais ce n'est pas un code préfixe.

Représenter des caractères et des textes

Les lettres ? Toutes des nombres !

Dans ce chapitre, nous voyons comment sont représentés les caractères et les textes de toutes les langues du monde.

Nous expliquons pourquoi il existe plusieurs codes tels *ASCII*, *latin-1*, *latin-2*, *UTF-32*, *UTF-8*. Nous présentons ensuite les formats enrichis qui permettent de décrire la forme des caractères et des textes, comme le font les logiciels de traitement de texte.

Un exemple de format enrichi est le langage HTML.

Nous nous intéressons, dans ce chapitre, à la représentation des textes, c'est-à-dire des suites de *caractères*, éventuellement enrichies d'informations typographiques.

La représentation des caractères

Puisqu'un texte est une suite de caractères, on commence par s'intéresser à la représentation des caractères, c'est-à-dire entre autres choses aux lettres minuscules et majuscules, aux chiffres, aux signes de ponctuation et aux symboles mathématiques. Pour représenter ces caractères, on attribue un nombre à chacun.

Le *code ASCII*, par exemple, attribue le nombre 65 à la lettre « A », le nombre 66 à la lettre « B », le nombre 97 à la lettre « a » et le nombre 98 à la lettre « b ». Il représente 95 caractères : les 26 lettres minuscules, les 26 lettres majuscules, les 10 chiffres, les 32 symboles ! " # $ % &' ( ) * + , – . / : ; < = > ? @ [\ ] ^ _ ` { | } ~ et 1 signe d'espace. Il représente aussi 33 autres *symboles de mise en page*, par exemple le retour chariot qui signale la fin de la ligne et le saut de page qui signale le passage à la page suivante. Le code ASCII représente donc $95 + 33 = 128 = 2^7$ caractères, par des nombres qui peuvent eux-mêmes être représentés en binaire par des mots de sept bits. Ils sont en fait représentés par des mots de huit bits, le premier étant toujours un zéro.

Le code ASCII était à l'origine conçu pour des textes écrits en anglais, comme l'indique son nom, *American Standard Code for Information Interchange*. Il n'est pas adapté pour représenter des textes écrits dans d'autres langues, même celles qui, comme le français, utilisent l'alphabet latin, car ces langues utilisent des accents, des cédilles et autres signes diacritiques. C'est pourquoi on a tout d'abord conçu une extension du code ASCII, le code *latin-1*, qui contient 191 caractères. Aux 128 caractères du code ASCII, qui sont représentés comme en ASCII, s'ajoutent les lettres « é », « É », « è », « ç », « æ », « ñ », « ö », etc. qui permettent de représenter les textes écrits dans la plupart des langues d'Europe de l'Ouest, même si, pour le français, le « œ » a été oublié. Il manque toutefois des lettres utilisées par les langues d'Europe de l'Est, si bien qu'un autre format, le code *latin-2*, a été proposé pour ces langues. Ensuite, pour représenter les textes écrits en grec, russe, chinois, japonais, coréen, etc., il a fallu proposer un format universel : *Unicode*. Unicode recense près de 110 000 caractères et associe un nom et un numéro à chacun. *A priori*, ce numéro se code sur 32 bits. Cependant, Unicode existe en plusieurs déclinaisons, parmi lesquelles *UTF-32*, dans laquelle chaque caractère est ainsi exprimé sur 32 bits, et *UTF-8*, dans laquelle les caractères les plus courants sont exprimés sur 8 bits et les moins courants sur 16, 32 ou 64 bits, utilisant une idée discutée en détail au chapitre 12 à propos de la notion de compression.

Le format UTF-8 a vocation à devenir le standard, mais il ne l'est pas encore : malgré les efforts des comités de normalisation, l'humanité n'a pas encore réussi à se doter d'un format universellement accepté, si bien qu'il est parfois nécessaire de traduire un texte d'UTF-8 en latin-1 ou de latin-2 en UTF-8. Quand cette traduction n'est pas bien faite, les caractères accentués sont remplacés par des caractères bizarres.

Cependant, tous ces formats reposent sur une même idée : associer un nombre, c'est-à-dire un mot binaire, à chaque caractère. Tous ces formats sont accessibles sur le Web.

La représentation des textes simples

Un texte étant une suite de caractères, on peut le représenter en écrivant les caractères les uns après les autres.

SAVOIR-FAIRE Trouver la représentation en ASCII binaire d'un texte

En utilisant une table, on cherche le code ASCII de chaque caractère. Puis on traduit chacun de ces nombres en représentation binaire.

Exercice 8.1 (avec corrigé)

Trouver la représentation binaire en ASCII du texte « Je pense, donc je suis. »

On cherche la table des codes ASCII sur le Web de manière à traduire le texte, caractère par caractère : 74, 101, 32, 112, 101, 110, 115, 101, 44, 32, 100, 111, 110, 99, 32, 106, 101, 32, 115, 117, 105, 115, 46. On exprime ensuite chacun de ces nombres en binaire sur huit bits :
01001010 01100101 00100000 01110000 01100101 01101110 01110011
01100101 00101100 00100000 01100100 01101111 01101110 01100011
00100000 01101010 01100101 00100000 01110011 01110101 01101001
01110011 00101110.

Exercice 8.2

Trouver la représentation binaire en ASCII du texte « Cet exercice est un peu fastidieux. »

SAVOIR-FAIRE Décoder un texte représenté en ASCII binaire

On découpe la suite de bits en octets, on traduit chaque octet en décimal, puis on cherche en utilisant une table, le caractère exprimé par chacun de ces nombres.

Exercice 8.3 (avec corrigé)

Trouver le texte représenté en ASCII binaire par la suite de bits 01000011001001110110010101110011011101000010000001100110011000010110001101101001011010001100101.

On commence par découper la suite de bits en octets : 01000011 00100111 01100101 01110011 01110100 00100000 01100110 01100001 01100011 01101001 01101100 01100101. Chaque octet représente un nombre entier : 67, 39, 101, 115, 116, 32, 102, 97, 99, 105, 108, 101. On cherche ensuite la table des codes ASCII en ligne de manière à traduire chacun de ces nombres en une lettre : « C'est facile ».

Exercice 8.4

Trouver le texte représenté en ASCII binaire par la suite de bits 00110000011101000110010101110100011101000010001.

Exercice 8.5

Traduire en ASCII binaire votre phrase préférée, par exemple : « Le commencement de toutes les sciences, c'est l'étonnement. » en oubliant les accents. Traduire ensuite cette phrase en UTF-8 avec les accents.

Exercice 8.6

Traduire une phrase en ASCII binaire, puis la passer à son voisin qui la décode.

Exercice 8.7

On suppose que les seize lettres qui suivent sont codées ainsi :

ء	آ	أ	ؤ	إ	ئ	ا	ث	د	ش	ف	ك	ل	ن	و	ي
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Décoder le message suivant : 1011 1100 1001 1111 0000 1010 1111 0111 1101 0110 0101 1111 0110 1100 1011 1110 1000, puis en se faisant éventuellement aider d'une personne qui lit l'arabe ou en utilisant le mécanisme de traduction de Google, déterminer s'il correspond à la phrase « tout en code binaire » ou « les lettres deviennent des nombres ».

Exercice 8.8

On considère l'alphabet de 32 signes constitué des 26 lettres de l'alphabet, de l'espace et de cinq symboles de ponctuation : la virgule, le point, le point-virgule, le point d'interrogation et le point d'exclamation. On représente les caractères de cet alphabet par un code binaire de cinq bits présenté dans le tableau suivant. Sur la dernière ligne, on a fait figurer la traduction décimale de chaque code binaire.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z		,	.	;	?	!
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**

1. Quel est le troisième bit du code de la lettre « P » ?
2. Décoder le message suivant : 00001011100110101001011101010010001.
3. Écrire son prénom avec ce code.
4. Quels sont les caractères qui commencent par deux 1 ?
5. Pourquoi n'y a t-il pas de caractères en minuscule dans cet alphabet ?
6. On considère l'opération qui consiste à transformer les 0 en 1 et vice-versa dans chaque caractère. Recopier et remplir le tableau ci-après qui, à chaque caractère, associe le caractère obtenu par cette transformation.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z		,	.	;	?	!
!	?																														

7. On considère l'opération qui consiste à échanger le premier bit avec le dernier, et le deuxième bit avec l'avant-dernier ; recopier et remplir le tableau ci-après qui, à chaque caractère, associe le caractère obtenu par cette transformation.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z		,	.	;	?	!
A	Q																														

ATTENTION **Utiliser un code standard**

Pour les exercices 8.7 et 8.8, on a inventé un code valable exclusivement pour la durée de l'exercice. Pour partager des informations, il ne faut jamais faire cela, mais utiliser un code standard, connu de tous. Sauf, bien sûr, si on veut garder le texte secret (voir le chapitre 12).

Exercice 8.9

Selon Pierre Lecomte du Noüy, 01001100 01100101 00100000 01100010 01110101 01110100 00100000 01100100 01100101 00100000 01101100 01100001 00100000 01110011 01100011 01101001 01100101 01101110 01100011 01100101 00100000 01100101 01110011 01110100 00100000 01100100 01100101 00100000 01110000 01110010 01100101 01110110 01101111 01101001 01110010 00101100 00100000 00100000 01101110 01101111 01101110 00100000 01100100 01100101 00100000 01100011 01101111 01101101 01110000 01110010 01100101 01101110 01100100 01110010 01100101. Êtes-vous d'accord avec lui ?

La représentation des textes enrichis

Les textes en ASCII ou en Unicode sont simplement des suites de caractères. Les *éditeurs de texte* sont les logiciels qui manipulent ces suites de caractères. Toutefois, quand on écrit un texte, on peut souhaiter lui donner une forme spéciale, plus jolie, plus lisible, comme le fait un imprimeur. On peut jouer sur la police de caractères – Times, Courier, etc. –, sur la taille des caractères – 11 points, 12 points, etc. –, sur leur forme – romain, italique, etc. –, leur graisse – maigre, gras, etc. On peut aussi souhaiter découper un texte en chapitres et mettre en valeur les titres des chapitres, etc. Or, les seules caractéristiques que l'on puisse exprimer avec un code comme le code ASCII, par exemple, sont la casse d'une lettre – minuscule ou majuscule – et le découpage en paragraphes, grâce au symbole retour chariot. Les *traitements de texte* sont les logiciels qui permettent ces mises en pages plus élaborées.

Ceci a amené à enrichir ces formats, de manière à :

1. *qualifier* certaines parties du texte, par exemple en mettant certaines parties en gras ou en italique,
2. structurer le texte en *divisions* : un texte n'est pas uniquement une suite de paragraphes, mais est hiérarchisé en parties, chapitres, sections, sous-sections, etc.
3. présenter certaines informations sous forme de listes et de tables,
4. permettre de faire *référence* à d'autres textes,
5. donner des informations sur le texte : son titre, son ou ses auteur(s), sa date de création, sa langue, des mots-clés utilisés pour le rechercher parmi plusieurs textes, etc. Ces informations **sur** le texte, et non **du** texte, sont appelées des *métadonnées*.

Toutes ces considérations sont, bien entendu, valables aussi bien pour les textes manuscrits ou imprimés que pour les textes traités par les ordinateurs.

L'un de ces formats enrichis, qui est utilisé en particulier pour écrire des pages web est appelé le format HTML. En HTML, pour mettre un passage en gras, on le délimite par les *balises* `<b>` et `</b>` et pour le mettre en italique, on le délimite par les balises `<i>` et `</i>`. Ainsi le texte :

```
Ma <i>première</i> page <b>web</b>
```

s'affiche dans un navigateur :

> Ma *première* page **web**

Comme les parenthèses, les balises vont par deux : on ouvre le passage à mettre en gras avec la balise `<b>` et on le ferme avec la balise `</b>`.

Une division du texte est délimitée par les balises `<div>` et `</div>`, ainsi le texte :

```
<div>Ma première page web
<div> comporte une première sous-division pour dire « Bonjour tout le monde ! »</
div>
<div> et une seconde qui finit par « À bientôt ! »</div>
</div>
```

s'affichera dans le navigateur en rendant ces divisions explicites.

Comme les parenthèses, les balises peuvent s'emboîter les unes dans les autres, mais pas se chevaucher.

On indique qu'un passage est un titre en le délimitant par les balises `<h1>` et `</h1>` et que c'est un sous-titre en le délimitant par les balises `<h2>` et `</h2>`.

De même, les autres structurations du texte comme les énumérations ou les tableaux sont exprimées par d'autres balises.

Quand on écrit un texte, il est fréquent de mentionner d'autres textes : par exemple, de parler dans une lettre d'un livre que l'on a lu. Dans le cas d'un texte manuscrit ou imprimé, on donne en général une référence du texte cité, par exemple le titre du livre et son auteur, afin que le lecteur puisse s'y référer s'il le souhaite. Quand on veut exprimer, dans une page web, une référence à une autre page, on peut faire mieux que simplement indiquer l'adresse de la page web en question (par exemple l'adresse http://fr.wikipedia.org/wiki/Hypertext_Markup_Language) ; on peut changer l'apparence du passage où l'on fait la référence, pour indiquer au lecteur que s'il clique sur ce passage, le navigateur affichera la page demandée. On utilise pour cela les balises `<a>` et `</a>` : on encadre la partie du texte à qualifier par ces deux balises et on indique à l'intérieur de la balise `<a>` l'adresse de la page référencée. Par exemple le texte :

```
Pour les détails sur le langage HTML, on pourra consulter <a href = "http://
fr.wikipedia.org/wiki/Hypertext_Markup_Language">la page <i>Hypertext Markup
Language de Wikipédia</i></a>.
```

qui affiche dans un navigateur :

> Pour les détails sur le langage HTML, on pourra consulter la page *Hypertext MarkupLanguage de Wikipédia*.

Si l'on clique sur le passage en bleu et souligné, le navigateur affiche la page dont l'adresse est http://fr.wikipedia.org/wiki/Hypertext_Markup_Language. Un tel passage sur lequel on peut cliquer pour accéder à une autre page s'appelle un *lien*, et un texte qui contient au moins un lien est un *hypertexte*.

Les balises `<body>` et `</body>` délimitent le texte à afficher dans le navigateur. On indique avant ces informations les méta-données relatives à la page : son titre, le format utilisé pour les lettres accentuées, etc.

Voici, au bout du compte, un exemple de texte au format HTML :

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8"></meta>
  <title>Un exemple</title>
</head>

<body>
  <h1>Un titre</h1>
  <h2>Un sous-titre</h2>
  <div><a href="http://www.wikipedia.org/">Un lien</a></div>
  <div><b>Un passage important</b></div>
</body>
</html>
```

L'en-tête situé entre les deux balises `<head>` et `</head>` indique d'une part que le texte est exprimé en UTF-8, c'est l'objet de la ligne :

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8"></meta>
```

et d'autre part que le titre de la page est `Un exemple`.

Le contenu est situé entre les balises `<body>` et `</body>`. On y retrouve les balises `<b>`, `</b>`, `<i>`, `</i>`, `<h1>`, `</h1>`, `<h2>`, `</h2>`, `<div>`, `</div>`, `<a>` et `</a>` que l'on a décrites. Dans un navigateur, le texte s'affiche ainsi.

SAVOIR-FAIRE Écrire une page en HTML

Écrire le texte contenu dans cette page. Structurer ce texte en divisions. Identifier les titres de parties, les passages à mettre en gras, en italique, etc. et les références vers d'autres pages. Ajouter les balises `<body>` et `</body>` autour du corps du texte, l'en-tête qui contient les méta-données et terminer avec les balises `<html>` et `</html>`.

Exercice 8.10 (avec corrigé)

Écrire une page HTML qui présente les différents projets informatiques des élèves d'une classe.

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8"></meta>
  <title>Projets Ada Lovelace</title>
</head>

<body>
  <h1>Les projets de la classe de TS1 du Lycée <a href = "http://
www.adalovelace.fr">Ada Lovelace</a></h1>
  <div>
   <a href="http://www.adalovelace.fr/informatique/ts1/projets/backgammon/
index.html">Un programme qui <b>joue aux Backgammon</b></a>
  </div>
  <div>
   <a href="http://www.adalovelace.fr/informatique/ts1/projets/compression/
index.html">Un programme qui <b>compresse des images</b></a>
  </div>
  <div><a href="http://www.adalovelace.fr/informatique/ts1/projets/montecarlo/
index.html">Un programme qui <b>calcule des intégrales</b> sans peine</a>
  </div>
</body>
</html>
```

Exercice 8.11

Écrire une page HTML qui présente la liste des concerts et des spectacles présentés dans un théâtre.

Exercice 8.12

Changer le texte HTML `Ma <i>première</i> page web` pour que le mot « première » apparaisse non en italique, mais en gras.

Exercice 8.13

Ce texte HTML est incorrect. Comment le corriger ?

```
Il faut <i>comprendre/i> le codage des objets numériques pour les maîtriser.
```

Exercice 8.14

Dans ce texte, vers quel site web pointe le lien ?

```
Votre compte bancaire présente une anomalie. Cliquer <a href="http://grosse-
arnaque.com">ici</a> pour avoir de l'aide.
```

Comment ce texte s'affiche-t-il dans un navigateur ? Quel est l'intérêt de regarder le source HTML de cette page avant de cliquer ?

Exercice 8.15

Donner le source HTML du texte suivant sachant que le texte en bleu et souligné est un lien vers la page http://www.monlivre.fr/page2 :

> On pourra consulter la page suivante.

Ai-je bien compris ?

- Comment représente-t-on un caractère ?
- Quelle est la différence entre le code ASCII et le format Unicode ?
- Quelle est la différence entre le format Unicode et le format HTML ?