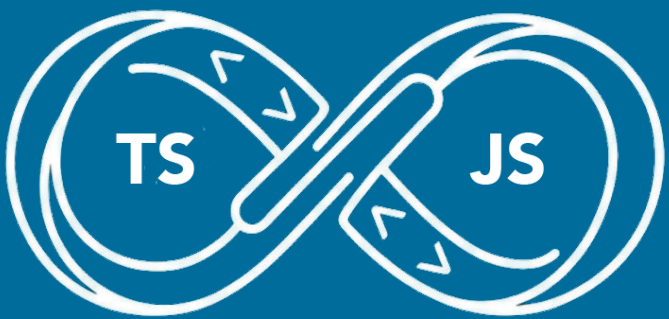
POUR LES FILIÈRES
BIOINFORMATIQUE ET SCIENCES SOCIALES

par Laurent Bloch

INITIATION à la PROGRAMMATION et aux ALGORITHMES

>TypeScript<

>JavaScript<



Editions TECHNIP

Laurent BLOCH
Conservatoire National des Arts et Métiers

INITIATION à la PROGRAMMATION et aux ALGORITHMES

>TypeScript<

>JavaScript<

2026
Editions TECHNIP 5 avenue de la République, 75011 PARIS

CHEZ LE MÊME ÉDITEUR

Stéphane Tufféry

- Data science, statistique et machine learning, 6e édition
- Modélisation prédictive et apprentissage statistique avec R, 3e édition
- Étude de cas en statistique décisionnelle, 2e édition
- Big data, machine learning et apprentissage profond

Société Française De Statistique

- Statistique des données fonctionnelles
- Données de composition
- Données manquantes
- Statistique et causalité

- Probabilités, analyse des données et statistique
  Gilbert Saporta
- Statistique et analyse des données avec R
  Gérald Quatrehomme, Philippe Du Jardin

Pour les filières bioinformatique et sciences sociales

par Laurent Bloch

- Initiation à la programmation et aux algorithmes avec **Scheme**
- Initiation à la programmation et aux algorithmes avec **Python**

Édition couverture et intérieur : Nathalie Loiseau
Image de couverture : Gemini

Dépôt légal : septembre 2026, imprimé en France
Site web de l'auteur : https://laurentbloch.net
ISBN 978-2-7108-1207-4

Table des matières

Avant-propos

« *Toute Pensée émet un Coup de Dés* »
Stéphane Mallarmé

Ce livre est issu d'un cours d'initiation à l'algorithmique et à la programmation, dispensé d'abord à l'Institut Pasteur, puis au Conservatoire national des arts et métiers, pour un public de biologistes. À cette fin, il développe des exemples orientés plus spécifiquement vers le traitement des textes et moins vers les applications numériques, qui conviennent mieux à des étudiants férus de mathématiques. Ce cours reposait sur l'emploi du langage Scheme, puis sur celui de Python, auxquels j'ai déjà consacré deux manuels. Ici les exemples reposent sur un couple de langages : TypeScript et JavaScript. Pourquoi deux langages, qui en fait n'en sont qu'un ? La réponse à cette question viendra dans quelques paragraphes.

L'algorithmique et la programmation constituent une branche majeure de la science informatique, à côté de l'architecture des ordinateurs, des systèmes d'exploitation et des réseaux, de l'intelligence artificielle, des bases de données. À l'issue d'un enseignement conforme à ce manuel, l'étudiant devrait pouvoir comprendre et analyser les résultats obtenus avec les programmes disponibles pour son domaine, et envisager de concevoir et de programmer de nouvelles méthodes. Il lui faudra sans doute, pour cela, apprendre de nouveaux langages. Mais, de toute façon, l'usage de l'ordinateur lui imposera de renouveler ses méthodes, tout au long de sa vie.

L'algorithmique est l'art de concevoir des algorithmes. Un algorithme est une idée de calcul, au sens large du mot : il peut désigner des opérations sur des nombres, mais aussi sur des expressions logiques, sur des textes, etc. Pour mériter le nom d'algorithme, cette idée doit recevoir une définition finie, composée d'étapes distinctes, dont chacune doit pouvoir être accomplie mécaniquement. Un bon exemple en est la méthode pour la multiplication « à la main », apprise à l'école élémentaire.

Comme toute idée, un algorithme peut être *faux*, comporter des erreurs logiques. Il existe aussi des problèmes sans solution algorithmique. Nous introduirons un langage de description d'algorithmes (dit *pseudo-code*) qui permet, sinon de garantir l'absence d'erreurs, du moins de les détecter plus facilement.

La programmation est l'art de traduire l'idée d'un algorithme en texte d'un programme, écrit dans un langage qu'un ordinateur va pouvoir interpréter. Outre les erreurs de conception de l'algorithme, le programme peut contenir d'autres types d'erreurs : infractions à la syntaxe du langage de programmation, violation des limites physiques de l'ordinateur utilisé, confusions dans les données soumises au programme...

Comme l'a écrit le professeur d'informatique au Collège de France, Xavier Leroy : « *Écrire un petit programme informatique est facile. Concevoir et réaliser un logiciel complet qui soit fiable, pérenne et résistant aux attaques reste extraordinairement difficile. C'est le but des sciences du logiciel que de concevoir et développer les principes, les formalismes mathématiques, les techniques empiriques et les outils informatiques nécessaires pour concevoir, programmer et vérifier des logiciels fiables et sécurisés.* » C'est pour progresser vers ce but que les informaticiens, depuis les années 1950, ont dépensé une part considérable de leur énergie à créer des langages de programmation plus commodes et plus abstraits, car moins liés aux caractéristiques physiques de l'ordinateur.

Une étape marquante de ces efforts a été franchie, en 1966, par Corrado Böhm et Giuseppe Jacopini [9], avec la démonstration du théorème de la programmation structurée, qui énonce que tout algorithme peut être programmé avec trois constructions simples (la séquence, l'alternative et la répétitive). Point n'est besoin d'y ajouter toutes les tournures baroques ou pittoresques issues de l'imagination d'auteurs facétieux.

Böhm et Jacopini ouvraient ainsi la voie aux méthodes de la *programmation structurée*, destinées à réduire les risques d'erreur de programmation. Dans cet avant-propos, nous résumerons ces méthodes par trois préceptes :

- la première qualité d'un programme est d'être lisible et intelligible par un être humain raisonnablement formé au langage utilisé ;
- pour ce faire, le texte du programme doit s'en tenir aux trois constructions de Böhm et de Jacopini ;
- enfin chaque donnée (variable ou constante) doit être *déclarée* avec un *type de donnée* (par exemple : nombre entier, chaîne de caractères, tableau de nombres...) afin d'éviter d'additionner des salades à des choux, comme nous l'enseignaient nos institutrices ou instituteurs d'antan.

Des langages de programmation conformes à ces principes ont été créés, et leur usage est recommandé. Mais il y a aussi des langages démagogiques qui, pour citer le professeur émérite au Collège de France, Gérard Berry, « *ont jeté par-dessus bord soixante ans de recherche en langages de programmation* ». Ces langages offrent l'illusion de la facilité, illusion qui débouche sur des erreurs difficiles à corriger. C'est le cas de Python, raison pour laquelle je le déconseille en dehors des usages pour lesquels il a été conçu, et qu'il accomplit très bien : l'écriture de scripts, c'est-à-dire de petits programmes de trois ou quatre lignes destinés à transmettre données et paramètres entre des programmes plus substantiels. La bibliothèque Biopython [8] est un bon exemple d'un tel usage, elle est

constituée de programmes classiques de la bioinformatique, écrits en C, coordonnés par des scripts en Python.

Voilà pourquoi ce livre concerne deux langages frères. Le premier, JavaScript, a été créé en 1996 par Brendan Eich de Netscape [1], initialement pour effectuer des actions sur des pages Web, telles que des animations ou des mises à jour de l'affichage, et pour en modifier le comportement. Assez vite, tous les navigateurs ont été dotés d'une machine virtuelle [2] (VM) JavaScript pour interpréter le langage, et très rapidement JavaScript est devenu immensément populaire. Des plates-formes telles que Node.js ou Deno ont été créées pour l'utiliser côté serveur, ou simplement comme un langage ordinaire. C'est aujourd'hui un des langages les plus populaires, pour le développement Web, mais pas seulement.

Cependant, JavaScript ne respecte pas la troisième règle de la programmation structurée – celle qui impose le typage des données – et cela, avec de sérieux inconvénients. En effet, JavaScript permet des mélanges de types particulièrement abracadabrants. Cela d'ailleurs viole la première règle, celle qui exige lisibilité et intelligibilité. C'est pourquoi des ingénieurs de Microsoft, sous la houlette de Anders Hejlsberg, ont créé, en 2012, une surcouche de typage baptisée TypeScript, dotée d'un trancompilateur qui traduit TypeScript en JavaScript.

Outre le typage des données et des fonctions, TypeScript permet la création de classes et d'interfaces, l'importation de modules – toutes choses absentes de JavaScript, et qui confèrent au langage une modularité recommandée par les idées de la programmation structurée. Le découpage d'un logiciel en fonctions et en modules, les plus indépendants que possible les uns des autres, suit le conseil de René Descartes dans le *Discours de la méthode* : « *diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre … conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu comme par degrés jusqu'à la connaissance des plus composés* ».

Ainsi, le programmeur peut développer son logiciel en TypeScript pour profiter de toutes les garanties de typage correct et de modularité, et le traduire facilement en JavaScript pour qu'il s'exécute sans difficulté, dans n'importe quel navigateur. Les programmes de ce livre seront donc introduits en TypeScript.

En résumé, je me suis efforcé de rester le plus simple et le plus sobre possible, et de me restreindre aux formes syntaxiques plus ou moins universelles, que l'étudiant retrouvera

1. À cette époque, Netscape et Sun Microsystems – où James Gosling et Patrick Naughton venaient de créer le langage Java – étaient étroitement associés, ce qui explique la similitude des noms des deux langages, bien qu'ils soient très différents l'un de l'autre.

2. Une machine virtuelle (VM) est un logiciel qui simule le fonctionnement d'un ordinateur. Une machine virtuelle JavaScript simule un ordinateur dont le langage machine serait JavaScript, ainsi elle peut interpréter et exécuter du code JavaScript. Le succès du langage tient en partie à sa simplicité : de ce fait la VM est légère et s'exécute sans encombre, même sur des téléphones un peu anciens.

sous une forme voisine dans les autres langages qu'il pourra être amené à utiliser. Enfin, j'ai partout préféré l'expressivité, qui facilite la compréhension, à la concision, qui peut rendre un texte de programme abscons.

La partie du livre consacrée aux bases de la programmation est suivie de l'exposé d'un certain nombre d'algorithmes utilisés en biologie. Il se trouve que ce ne sont pas des algorithmes numériques, mais des algorithmes du texte : tri, recherche, comparaison, similarité. À ce titre, ils peuvent aussi intéresser des littéraires.

Remerciements et détails pratiques

Le cours, à l'origine de ce livre, a été élaboré avec William Saurin, auquel nous devons de nombreux développements et algorithmes.

Tous les programmes de ce livre, que le lecteur est invité à reproduire, modifier et améliorer pour son propre compte, ont été réalisés et testés. Je ne saurais trop conseiller à qui envisage sérieusement de travailler en informatique d'adopter, dès maintenant, GNU/Linux (ou un autre système d'exploitation libre de la famille Unix, tel que FreeBSD, OpenBSD ou NetBSD). Les autres systèmes sont destinés à toutes sortes d'usages, mais pas à l'informatique.

Ce livre a été écrit et composé au moyen de logiciels libres (GNU Emacs, Linux, TEX, LuaLATEX, BibLATEX, `xfig`, Xindy) et, pour la typographie française, le style e-french de Bernard Gaulle. Il convient d'en remercier ici les auteurs et contributeurs, dont le travail désintéressé élargit le champ de la liberté d'expression pour tous.

Chapitre 1. Fondations

Sommaire

> « *La programmation est une transmission d'un raisonnement à une machine, capable de le reproduire. Par cette reproduction, sans laquelle il n'est point de science, l'ordinateur s'affirme comme un instrument d'objectivation du raisonnement. Il s'inscrit dans la lignée des appareils qui ont permis de passer de l'expérience empirique à l'expérimentation scientifique: l'informatique se distingue des mathématiques par un outil d'expérimentation permettant d'observer, vérifier, réfuter un raisonnement.* »
>
> Emmanuel Saint-James [42]

1.1 Calcul automatique

Le propos de cet ouvrage est d'apprendre à programmer les ordinateurs. N'importe quel ordinateur, puisqu'ils sont tous équivalents, et pour tout programme, puisque le propre d'un ordinateur est l'universalité, c'est-à-dire la capacité à traiter tous les problèmes traitables effectivement.

Avant de commencer, il convient de faire un tour d'horizon sommaire des questions mises en jeu par ces premiers mots.

L'informatique moderne est née de la recherche, entreprise au début du XX^e siècle par Bertrand Russel et Alfred North Whitehead et publiée sous le titre *Principia Mathematica*, pour constituer les mathématiques en un système formel [1] où toute proposition pourrait être démontrée par un calcul logique. David Hilbert et Kurt Gödel accomplissent des pas décisifs dans l'exploration de ce programme. En 1931 Gödel démontre que pour tout système formel suffisant pour représenter l'arithmétique il existe des propriétés vraies dont on ne peut démontrer la vérité ni la fausseté avec les règles du système lui-même. Le théorème de Gödel ruine le rêve de réunir les mathématiques dans un système déductif parfaitement cohérent qui éliminerait tout recours à l'intuition. Mais de l'effervescence intellectuelle autour du projet des *Principia* vont sortir les idées fondatrices de l'informatique [2].

En 1936 Alan Turing, à la suite de Gödel, s'attaque à un problème de décidabilité : un système est décidable s'il existe une procédure effective pour distinguer les propositions démontrables des autres. Pour définir plus rigoureusement la notion de procédure effective, Turing élabore un modèle d'automate appelé par la suite *machine de Turing*, qui lui permet de préciser la notion d'exécution d'un algorithme (cf. p. 211 une brève description du principe de la machine de Turing).

Inventer des procédures effectives (des algorithmes) consiste à déterminer un enchaînement d'opérations élémentaires qui exécuteront les calculs nécessaires à la solution de problèmes pour lesquels existent des solutions calculables. Ce livre expose l'art de réaliser des algorithmes. Turing montre que son modèle de calcul est universel, c'est-à-dire que toutes les machines de Turing sont équivalentes. Il formule la thèse selon laquelle tout algorithme est calculable par une machine de Turing. Ces idées fondent la théorie de la programmation des ordinateurs. Il revient à von Neumann de concevoir, en 1945, l'architecture générale des appareils concrets qui vont réaliser les calculs selon le modèle de Turing, architecture si élégante que les ordinateurs d'aujourd'hui sont encore construits, pour l'essentiel, selon ses principes.

1.2 Informatique

Le mot *informatique* combine *information* et *automatique* :

– l'information est ce qui donne à quelque chose, de l'intérieur de cette chose, une forme ;

1. Un *système formel* est constitué de règles en nombre fini opérant sur des ensembles dénombrables.

2. Sur toutes ces questions et leurs implications pour l'informatique moderne, on lira avec profit l'ouvrage de Gilles Dowek, *Les Métamorphoses du calcul* [19].

– l'automatique permet l'automatisation de ce processus ;
– le résultat est un automate.

La chose, à laquelle il s'agit ici de donner une forme, est une idée, que nous nommerons un *algorithme*. La forme donnée sera appelée un *programme*, qui est un texte écrit selon un *langage*.

L'écriture d'un programme est une activité créatrice, elle suppose la maîtrise d'un langage.

Un langage de programmation permet d'écrire des programmes destinés à être exécutés par une machine, un ordinateur [3].

Un ordinateur est un calculateur automatique universel programmable. Universel, parce qu'il peut exécuter tout programme qui réalise n'importe quel algorithme (il y a des problèmes pour lesquels il n'y a pas d'algorithme connu : en d'autres termes, il y a des problèmes sans solution, et des solutions incalculables). Automatique, parce qu'il peut enchaîner un nombre quelconque d'opérations sans intervention humaine. Programmable, parce qu'il comporte des dispositifs de commande destinés à enchaîner ces opérations dans un ordre prescrit, qui peut changer au fur et à mesure des résultats intermédiaires.

L'ordinateur est une machine à états : un calcul (au sens large, non restreint aux opérations numériques) consiste à partir d'un état de départ, qui contient les données du calcul, pour atteindre un état final, qui contient les résultats.

Les états successifs du calcul sont enregistrés dans un dispositif appelé *mémoire* [4]. L'ordinateur contient aussi une *unité arithmétique et logique* (UAL), constituée de circuits électroniques qui implémentent des *instructions*, capables de modifier le contenu de la mémoire, et ainsi de passer d'un état initial à un état final qui donne les résultats du calcul. L'UAL comprend également des registres, petits éléments de mémoire destinés à accueillir temporairement les données sur lesquelles porte le calcul en cours. Pour finir, l'*unité de contrôle* [5] pilote l'enchaînement des instructions.

Un programme est un texte qui décrit les instructions à exécuter, les zones de la mémoire qu'elles doivent affecter, et la façon dont l'unité de contrôle doit les enchaîner. Les instructions matérielles, réalisées par les circuits électroniques de l'UAL et de l'unité de contrôle de l'ordinateur, sont appelées instructions primitives, ou simplement *primitives*. Il existe des langages de programmation – comme celui utilisé dans ce livre

3. Ces programmes servent également, et ce n'est pas le moindre de leurs usages, à transmettre des raisonnements à des lecteurs. Apprendre à programmer n'est pas seulement apprendre à actionner des ordinateurs, mais aussi apprendre à raisonner sur des programmes.

4. Ce terme est d'un anthropomorphisme malheureux, les anglophones utilisent souvent *storage*, plus modeste et plus exact.

5. Une traduction plus exacte de l'anglais *control unit* serait « unité de commande ». L'usage a entériné l'impropriété.

– qui fournissent des instructions plus faciles à utiliser ques les primitives. Mais, pour être effectuées, ces instructions doivent être traduites en instructions primitives par les programmes de traduction (compilateurs ou interpréteurs), que nous évoquerons à la section D.5, p. 234.

La mémoire de l'ordinateur (c'est l'idée fondamentale de von Neumann, qui a donné son nom à cette architecture) contient des informations de deux types : des programmes et des données. Les programmes et les données sont représentés avec les mêmes symboles, seule la sémantique permet d'interpréter leurs textes respectifs. D'ailleurs, le texte d'un programme peut parfois être envisagé comme des données pour un autre programme, par exemple pour un programme de traduction d'un langage dans un autre.

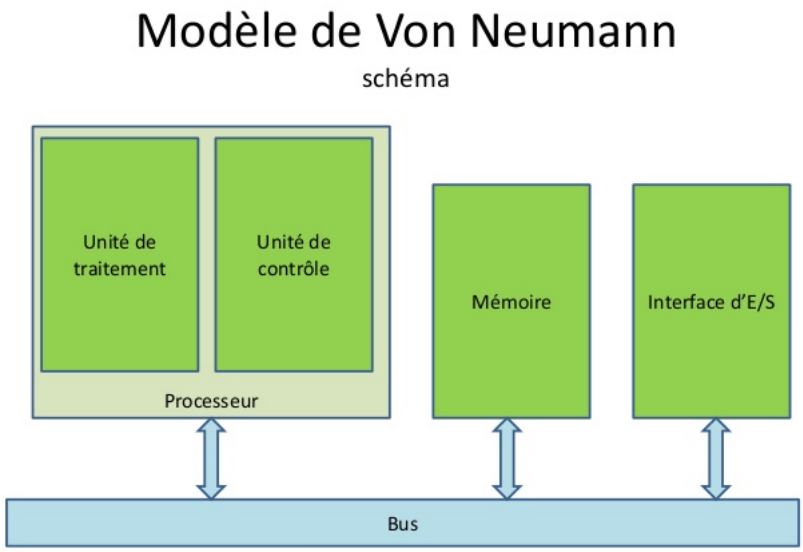


Figure 1.1 : *Architecture de von Neumann.*

La figure 1.1 donne le schéma de l'architecture de von Neumann (l'UAL y est nommée « unité de traitement »). Si Turing a fondé la science informatique, von Neumann a inventé l'ordinateur, et Newmann et Wilkes ont construit les deux premiers. On trouvera à l'annexe F, p. 241, une discussion autour de ces questions : qui a inventé l'ordinateur, et qui a construit le premier ?

1.3 Machine

L'architecture que von Neumann propose pour l'ordinateur est décrite par la figure 1.1.

Les unités de contrôle[6], arithmétique et d'entrée-sortie constituent à elles trois l'unité centrale, ou le *processeur* de l'ordinateur. Le processeur est composé de circuits électroniques qui peuvent exécuter des actions (des calculs au sens large). L'ensemble des actions « câblées » dans le processeur constitue le jeu d'*instructions* du processeur et détermine le *langage* élémentaire de son utilisation, appelé « langage machine ». L'annexe D p. 231 présente un exemple très simplifié de langage machine et de programme dans ce langage.

Le rôle de l'unité de contrôle est de permettre le déclenchement de l'action voulue, décrite par le texte d'une instruction, au moment voulu. Cette instruction peut appartenir à l'unité arithmétique, à l'unité d'entrée-sortie ou à l'unité de contrôle elle-même. Une instruction peut, en outre, consulter le contenu de la *mémoire* (la « lire ») ou le modifier (y « écrire »). De façon générale, une action consiste soit à consulter ou à modifier l'état de la mémoire ou d'un des registres A ou R (qui sont des éléments de mémoire spéciaux incorporés à l'unité centrale, l'accumulateur A est un registre comme R), soit à déclencher une opération d'entrée-sortie (communication avec le monde extérieur, et notamment l'utilisateur humain).

Comment indique-t-on à l'unité de contrôle le « moment voulu » pour déclencher telle ou telle action ? Par le texte d'un *programme*, lui-même constitué d'instructions qui appartiennent au *langage* du processeur. Où est le programme ? Dans la *mémoire*.

Le chemin par lequel unité centrale, mémoire et organes d'entrée-sortie communiquent s'appelle, de façon générique, un « bus ». Plus formellement, un bus est un graphe connexe complet : cela signifie, en langage courant, que tous les éléments connectés au bus peuvent communiquer entre eux.

1.4 Information

La transmission de l'information par les réseaux ainsi que son traitement par les ordinateurs nécessitent qu'elle soit représentée selon des conventions précises, et enregistrée dans des mémoires. On appelle *codage* l'opération qui consiste à figurer une information sous sa forme codée. La mémoire est un objet matériel susceptible de prendre différents états[7]. Un élément de base de la mémoire peut prendre deux états distincts et désigner

6. Une traduction plus exacte de l'anglais *control unit* serait « unité de commande ». L'usage a entériné l'impropriété.

7. Les registres peuvent être considérés ici comme des parties de la mémoire.

ainsi une *information* élémentaire, ou *bit* (*binary digit,* chiffre binaire). Cette représentation d'une information par un élément de mémoire s'appelle un *code*. Une mémoire avec beaucoup de bits permet le codage d'informations complexes, dans la limite de la taille de la mémoire. Par commodité, on manipule souvent les bits par groupes de 8, les *octets*.

1.4.1 Codage

Le codage fait correspondre des groupes de bits (le plus souvent, un ou plusieurs octets) à des symboles. Les symboles les plus simples sont les chiffres et les lettres. Pour représenter des informations complexes, on peut définir des méthodes et des règles pour regrouper des symboles, puis associer un élément d'information à un groupe de symboles construit selon les règles. On appellera *langage* un ensemble de symboles ou de groupes de symboles, construits selon certaines règles, qui sont les *mots* du langage. La *syntaxe* du langage est l'ensemble des règles d'agencement des mots du langage. La signification (ou encore la *sémantique*) d'un mot d'un langage est affaire d'interprétation, et peut dépendre du contexte. Un ensemble de mots peut constituer un texte, généralement contenu dans un fichier enregistré sur un support physique tel que disque dur, clé USB, etc.

Les numéros d'immatriculation des automobiles, les numéros d'inscription au répertoire des personnes physiques (NIR, dits faussement numéros de sécurité sociale), le langage machine d'un ordinateur sont des langages[8].

Pour être communiqué à un utilisateur (lecteur, auditeur, spectateur), un texte enregistré dans un fichier devra être interprété par un logiciel adapté au langage dans lequel il a été rédigé : logiciel de traitement de texte s'il est destiné à être imprimé, logiciel de traitement d'image s'il s'agit d'un document graphique, logiciel de traitement du son s'il s'agit d'un enregistrement musical numérisé. Ainsi, l'examen direct des octets d'un fichier musical ne nous évoquera pas sa mélodie, il faut pour cela un logiciel spécial de traduction de ces octets en signaux envoyés à un dispositif physique de restitution du son.

La mémoire de l'ordinateur contient des informations de deux types : des programmes et des données. Les programmes et les données sont représentés avec les mêmes symboles, seule la sémantique permet d'interpréter leurs textes respectifs (c'est l'idée fondamentale de von Neumann). D'ailleurs, le texte d'un programme peut parfois être envisagé comme des données pour un autre programme, par exemple un programme de traduction d'un langage dans un autre. Ainsi nous verrons ci-dessous comment le texte d'un programme TypeScript sera traduit en texte d'un programme JavaScript par un programme spécial, le compilateur TypeScript. Ensuite, ce programme JavaScript sera

8. La suite nous réserve un raffinement plus grand de la notion de langage.

traduit en langage machine par la machine virtuelle (VM) JavaScript du navigateur utilisé.

1.4.2 Codage de l'écriture

Un exemple intéressant de codage de l'information est la représentation des signes de l'écriture, caractères alphabétiques, sinogrammes, hiéroglyphes, etc., désignés par le terme générique *caractère*.

Dans les débuts du traitement des textes par ordinateur, il existait deux codes rivaux : ASCII *(American Standard Code for Information Interchange)* et EBCDIC *(Extended Binary Coded Decimal Interchange Code)*. Le code ASCII utilisait 7 bits, EBCDIC 8 bits, ils ne pouvaient désigner que les caractères de l'alphabet latin (sans accents, cédilles, trémas, ni autres *umlauts*), les chiffres, les signes de ponctuation et quelques autres symboles. Ainsi, le chiffre 1 était codé 0110001 en ASCII et 11110001 en EBCDIC, et pour la lettre A, on avait respectivement 1000001 et 11000001, etc.

En 1991 apparaît une norme internationale, Unicode, constituée d'un répertoire de 137 929 caractères, qui permet de désigner chaque caractère de toutes les écritures passées et présentes[9] par un nom et un code numérique, nommé *point de code (codepoint)*.

Il existe plusieurs normes de codage informatique pour représenter les caractères Unicode, la plus répandue est UTF-8. Il est bon de savoir que les codes UTF-8 sont de taille variable, ils comportent au minimum 8 bits (d'où le nom de la norme), mais peuvent en compter jusqu'à 32[10]. Ainsi :

Nom Unicode du caractère	Point de code	UTF-8 (décimal)	UTF-8 (hexadécimal)
Right single quotation mark ’	U+2019	226 128 153	e28099
Apostrophe '	U+0027	39	27
Latin small letter m	U+006D	109	6d
Latin capital letter E with grave È	U+00C8	195 136	c388

Le caractère *Right single quotation mark* est utilisé pour noter l'apostrophe latine, en français par exemple, de façon plus élégante que par l'apostrophe de l'anglais, qui s'accommode bien de sa laideur parce qu'il l'utilise fort peu.

9. La moitié de la table de codage est dévolue aux caractères de la langue chinoise, qui existent chacun en deux versions : la version traditionnelle en usage à Taïwan et la version dite simplifiée, utilisée en Chine continentale.

10. Cf. la table complète : `https://www.fileformat.info/info/charset/UTF-8/list.htm`

Pour la notation décimale des codes, la valeur binaire de chaque octet est convertie en base 10, ce qui donne un nombre compris entre 0 et 255, ainsi par exemple pour È :

$$(\mathrm{c3})_{16} = (11000011)_2 = (195)_{10}$$

$$(88)_{16} = (10001000)_2 = (136)_{10}$$

Le code hexadécimal c388 est fourni par la table de caractères, le code UTF-8 de È est donc $(1100001110001000)_2$. C'est parce que 16 est une puissance de 2 que le code hexadécimal est obtenu simplement par la concaténation des conversions en base 16 de chaque octet. La notation hexadécimale est expliquée plus en détail p. 223.

Rien dans le texte binaire ne permet de détecter quel codage est utilisé, cette information doit être fournie explicitement par l'auteur du document (ou par son logiciel).

Ce vaste sujet excède les limites de ce livre, nous ne pouvons que renvoyer le lecteur aux excellents articles de Wikipédia Unicode, UTF-8 et Glyphe.

Les quatre concepts fondamentaux de l'informatique

Ce bref exposé liminaire nous a permis de nommer les quatre concepts fondamentaux de l'informatique :

- machine ;
- langage ;
- algorithme ;
- information.

Cette articulation autour de quatre concepts est le fruit des réflexions d'universitaires et d'enseignants des groupes EPI (Association Enseignement public & informatique) et ITIC (Informatique et TIC).

1.4.3 L'information sous l'angle formel

Nous ne saurions omettre la mention d'une théorie formelle de l'information [11], même si le lecteur rebuté par les formules doit savoir qu'il pourra programmer toute sa vie en l'ignorant.

C'est en pensant aux systèmes de télécommunications que Henrik Nyquist et Claude Shannon ont élaboré cette théorie. Mais des recherches ultérieures – notamment celles de von Neumann [37] – en ont élargi la portée, en la rattachant à la physique statistique par la notion d'entropie. Cette généralisation, qui stipule que toute interaction entre deux systèmes peut être interprétée comme un signal porteur de message, fait de la théorie de l'information une théorie fondamentale, au même titre que la théorie de la relativité ou la mécanique quantique.

1.4.4 Information et probabilités

Le transfert d'information dans un système de communication s'effectue par *messages*. Un message est une suite de signes (de symboles) tirés d'un *alphabet*. L'ensemble $S = \{m_1 ... m_i ... m_n\}$ des messages que l'on peut former à l'aide d'un alphabet donné constitue une *source (discrète) de messages* : un *texte* est une suite de messages.

La notion d'information est liée à l'ignorance du destinataire quant aux messages émis depuis S : il n'y a apport d'information que si le destinataire ignore le contenu du message qu'il va recevoir. L'incertitude, quant à la teneur du message, est d'autant plus grande que le message a une faible probabilité d'être émis ; inversement, la réception de ce message contribue à lever une incertitude encore plus grande, et apporte donc une plus grande quantité d'information. Ainsi apparaît une relation entre la quantité d'information d'un message m_i, et sa probabilité d'émission p_i, relation représentée par la fonction logarithmique suivante :

$$I(m_i) = \log_\alpha(1/p_i) = -\log_\alpha p_i$$

I étant l'incertitude, ou l'information, α le nombre de symboles de l'alphabet utilisé. On en déduit immédiatement que la valeur de l'unité d'information en base α est celle d'un message de probabilité $\frac{1}{\alpha}$. On prend généralement $\alpha = 2$, l'unité correspondante étant le bit.

11. Michel Volle me fait remarquer que cette expression consacrée par l'usage est malheureuse, il serait plus approprié de dire « théorie de la communication », conformément d'ailleurs au titre de l'article de Shannon, *A mathematical theory of communication*, (Bell System Technical Journal, vol. 27, juillet et octobre 1948) [44, 43], ou « théorie des données ». L'ordinateur, ou le réseau de communication, contiennent et transmettent des données, qui ne deviennent de l'information que dans l'esprit d'un être humain, capable (et lui seul en est capable) de leur donner un *sens*. Voir à ce sujet `http://www.volle.com/ulb/021115/textes/vocabulaire.htm`.

La fonction logarithme

Le logarithme de base α de x, noté $\log_\alpha x$, est le nombre m tel que $\alpha^m = x$. On vérifie que $\log_\alpha \alpha = 1$ et que, quelle que soit la valeur de α, $\log 1 = 0$ et $\log(a \times b) = \log a + \log b$. Cette dernière propriété est très intéressante puisqu'elle permet, avec une table de logarithmes, de faire des multiplications même si on ne connaît que la table d'addition. Elle reste aussi le principe de base de la règle à calcul, qui n'est autre qu'une table de logs en plastique. De même, $\log(a^n) = n \log a$.

Le logarithme népérien est celui qui utilise le nombre e comme base, il est la fonction réciproque de la fonction exponentielle. Pour les applications pratiques, la base 10 est plus commode. Le passage d'une base à une autre est simple : $\log_\beta x = \frac{\log_\alpha x}{\log_\alpha \beta}$.

Nous retrouverons les logarithmes à la section B.5 de l'annexe B, p. 220, pour calculer le nombre de chiffres d'un nombre.

Pour un contenu plus facile à retenir que la formule ci-dessus, on peut utiliser les logarithmes décimaux, et regarder la quantité d'information transmise par un message émis avec une certaine probabilité :

probabilité d'émission	quantité d'information
$p_1 = 1$	$I(m_1) = \log_{10}(1/p_1) = -\log_{10} 1 = 0$
$p_2 = 0{,}1$	$I(m_2) = \log_{10}(1/p_2) = -\log_{10} 10^{-1} = +1$
$p_3 = 0{,}01$	$I(m_3) = \log_{10}(1/p_3) = -\log_{10} 10^{-2} = +2$
...	...

Cette définition probabiliste de la quantité d'information d'un message montre qu'elle dépend avant tout de la source de messages utilisée ; cette dernière peut être caractérisée par une quantité d'information (ou incertitude) moyenne, d'après l'expression :

$$H(S) = -\sum_{i=1}^{n} p_i \times \log_\alpha p_i$$

Celle-ci permet d'évaluer *a priori* la quantité moyenne d'information que peut fournir un message ; sa valeur est maximale pour des messages équiprobables. Cette grandeur a la même forme que l'entropie thermodynamique, on l'appelle entropie de S.

L'entropie permet d'évaluer la richesse informationnelle d'un texte : Shannon a montré que, si l'information moyenne d'un alphabet de 27 signes équiprobables était

$\log_2 27 = 4{,}75$ bits/lettre, le contenu d'un texte anglais ordinaire n'était que de 1 bit/lettre, soit une redondance de : $1 - \frac{1}{4{,}75}$, ou 80% de signes inutiles. Une équipe de l'université de Lyon a établi que le débit d'information était identique dans toutes les langues, quelle que soit la vitesse de locution : 39 bits/seconde. Les langues au débit rapide, comme le japonais, utilisent davantage de syllabes pour transmettre la même quantité d'information que celles au débit plus lent, comme le vietnamien. Des calculs de forme analogue sont effectués par les logiciels d'analyse de séquences biologiques.

1.5 Algorithme

> *Ces lignes (et quelques autres) doivent beaucoup à l'ouvrage de Thérèse Accart Hardin et de Véronique Donzeau-Gouge Viguié* [1]*, dont la lecture ne saurait être trop recommandée au lecteur avide d'explications claires sur des idées complexes.*

L'informatique est la science du traitement de l'information. Nous avons exposé une idée sommaire de la nature de l'information. Qu'est son traitement ? C'est, en partant d'informations connues appelées *données*, faire élaborer par un agent exécutant des informations nouvelles appelées *résultats*. Un traitement est l'application d'une méthode de passage d'un ensemble de données D à un ensemble de résultats R : à toute donnée d de D on fait correspondre un élément r de R. Cette définition est inspirée de la notion mathématique de fonction.

La méthode de passage invoquée doit être adaptée à l'agent exécutant. Par exemple, pour l'opération « somme de deux nombres entiers », la façon d'obtenir à partir d'un couple de nombres leur somme sera décrite différemment selon l'agent exécutant, c'est-à-dire un être humain muni d'un crayon et de papier ou le processeur d'un ordinateur.

Le traitement des données doit être *effectif*, c'est-à-dire que la méthode de passage doit pouvoir aboutir pratiquement au résultat. Prenons un exemple : soit D l'ensemble des numéros d'immatriculation [12] des voitures immatriculées en France. Les symboles utilisés sont des chiffres et des lettres. Les règles de formation des numéros d'immatriculation sont la syntaxe du langage. La sémantique d'un numéro est l'identité de la voiture qui le porte. Considérons R, l'ensemble des départements français. La correspondance de tout numéro d'immatriculation bien formé au département d'immatriculation de la voiture qui le porte est un traitement de D, on sait comment le réaliser. En revanche, la correspondance de tout numéro d'immatriculation bien formé à la commune d'immatriculation de la voiture associée n'est pas un traitement de D, car on ne sait pas élaborer

12. On considère ici les *anciens* numéros d'immatriculation, parce que les nouveaux sont, à juste titre, dépourvus de sémantique, ce qui est préférable pour un identifiant, mais regrettable pour les enseignements d'informatique.

cette information à partir du numéro, bien qu'elle existe sûrement. Cette correspondance ne peut pas être décrite de manière effective, c'est-à-dire par un *algorithme*.

Un algorithme est la description, pour un exécutant donné, d'une méthode de résolution d'un problème, autrement dit une suite d'opérations qui fournit le résultat cherché.

La description de la méthode, c'est-à-dire l'algorithme, doit être adaptée à l'exécutant chargé d'effectuer le traitement. L'exécutant sait effectuer un nombre fini d'actions, que l'on nomme ses primitives (ce sont, par exemple, les instructions du jeu d'instructions du processeur décrit ci-dessus). L'algorithme doit être constitué d'une combinaison de ces primitives. Pour construire toutes les combinaisons possibles de primitives (et donc, pour programmer tous les algorithmes possibles), Corrado Böhm et Giuseppe Jacopini ont démontré dans un article célèbre des CACM, en mai 1966, intitulé "*Flow diagrams, Turing machines and languages with only two formation rules*" [9], qu'il suffisait de savoir réaliser trois combinaisons élémentaires :

- l'enchaînement de deux primitives (effectuer une action à la suite d'une autre), soit une *séquence* ;
- la répétition d'une primitive donnée, soit une *répétitive* ;
- et le choix, pendant l'exécution d'un traitement, entre deux primitives selon le résultat d'un test, soit une *alternative*.

Un algorithme fournit, pour un exécutant donné, la décomposition de la tâche à réaliser en primitives de l'exécutant.

1.6 Programme

À présent, si nous résumons notre description introductive et simplifiée de ce qu'est un ordinateur, nous dirons qu'il dispose d'un processeur capable d'accomplir des actions dites primitives, de les enchaîner dans l'ordre voulu, de les répéter, et surtout de choisir – en fonction du résultat des actions précédentes – la prochaine action à effectuer entre deux ou plusieurs possibilités (ces cas possibles sont en nombre fini et connus à l'avance).

Notre ordinateur dispose aussi d'une mémoire qui contient des informations codées, qui sont de deux types : des programmes et des données. Pour l'instant, nous nous contenterons de dire que des conventions sémantiques relatives au contexte de démarrage de l'ordinateur permettent de distinguer le programme des données à traiter : lorsqu'il commence à agir, le processeur va par convention aller chercher dans la mémoire, à un emplacement convenu, le texte de la première instruction à exécuter. Les règles d'enchaînement du langage assurent, en principe, le déroulement cohérent de la suite.

Le programme est le texte de l'algorithme dont l'exécution est désirée, rédigé dans le langage machine de l'ordinateur. Ce texte énumère les primitives de l'ordinateur dont

l'exécution mènera à la production du résultat recherché. Le programme est dit juste s'il se termine [13].

1.7 Langage

Fondamentalement, un ordinateur est une machine qui effectue les traitements décrits par des algorithmes rédigés dans son langage machine. Il existe, heureusement, beaucoup d'autres langages de programmation.

Un langage de programmation est un système de notation, au moyen duquel un être humain peut transmettre un algorithme à un ordinateur ou à un autre être humain. Toute une gradation existe entre les langages bien adaptés aux ordinateurs mais difficiles à comprendre pour les personnes, et les langages proches de notations habituelles aux personnes mais difficiles à faire interpréter par un ordinateur. Chaque ordinateur possède un langage qui lui est intrinsèque, son langage machine, et, pour être exécutés, les programmes écrits dans des langages plus commodes doivent être traduits en langage machine. Cette traduction est réalisée par des programmes spéciaux, dont il existe deux grandes catégories : les *compilateurs* et les *interprètes*.

Le premier problème posé par l'usage des ordinateurs fut de communiquer des algorithmes à la machine. À l'origine, le seul moyen disponible était la programmation directe en langage machine : chaque instruction codée en binaire correspondait (et correspond d'ailleurs toujours) à un circuit logique de l'unité centrale, et les données étaient désignées par leur adresse, c'est-à-dire le rang de la « case » mémoire les contenant. Dès l'EDSAC (le premier ordinateur, en 1949), fut employé un langage permettant de désigner les instructions par un code alphabétique mnémonique et d'utiliser des adresses symboliques (alphabétiques) au lieu des adresses physiques : ce type de langage est appelé un *assembleur*. Il faut un programme de traduction (ou assembleur) pour le traduire en langage machine, et notamment, pour traduire les symboles en adresses (physiques). Ce programme est en fait un compilateur particulièrement simple, parce que le langage à traduire est très proche du langage machine

Mais, progressivement, vint le besoin de langages moins dépendants de la structure de l'ordinateur, et plus adaptés à la description des problèmes à traiter – que l'on appellera les langages évolués, traduits en langage machine par des compilateurs ou des interprètes, et qui sont des métalangages par rapport à l'assembleur.

13. On souhaite aussi généralement qu'il donne en se terminant un résultat pertinent. La caractérisation de la justesse d'un programme par le fait qu'il se termine peut choquer le lecteur. C'est néanmoins une condition logique primordiale, susceptible de démonstration formelle, d'où son importance. Une fois qu'elle est satisfaite, il est loisible de formuler des exigences sur la relation qui peut exister entre le domaine des données D et celui des résultats R.

Les premiers langages évolués furent sont des langages dits « impératifs », fondés sur la notion d'état de la mémoire. Ces langages, inspirés du modèle de von Neumann, comportent, comme le langage machine, des instructions qui produisent des modifications de la mémoire (instruction d'affectation). La rédaction d'un programme en langage impératif consiste à écrire la suite des instructions qui vont causer les états successifs par lesquels devra passer la mémoire, pour qu'en partant d'un état initial permettant l'initialisation du programme, elle arrive dans un état final fournissant les résultats recherchés. Un tel programme explicite *comment réaliser* le traitement voulu par le programmeur.

Une autre approche pour concevoir des langages de programmation est de s'inspirer du modèle de la machine de Turing, dont on trouvera une description sommaire à l'annexe A, p. 211. Ces langages reposent sur la notion de fonction, et sont donc appelés langages fonctionnels [14]. Ils ne comportent pas d'instructions, mais le calcul est exprimé par des fonctions qui, appliquées à des arguments, rendent le résultat recherché. Des programmes rédigés avec de tels langages explicitent *quoi réaliser* pour obtenir le résultat voulu.

14. Le terme « fonctionnel » suggère « programmation avec des fonctions ». Certains pensent que l'adjectif « applicatif » qualifierait mieux ces langages. Sans doute. Mais « fonctionnel » est entré dans l'usage.

INITIATION à la PROGRAMMATION et aux ALGORITHMES

>TypeScript< >JavaScript<

L'informatique et les algorithmes sont aujourd'hui présents dans tous les domaines. La programmation est matière d'enseignement obligatoire au lycée, et **la biologie comme les sciences sociales ne se conçoivent plus désormais sans un appui informatique. Par ailleurs, sur le marché de l'emploi, la préférence va aux spécialistes qui ont une double compétence** en informatique.

Conçu à l'origine autour d'exemples biologiques (comme l'analyse de séquences), cet ouvrage constitue une véritable initiation à la programmation. Les méthodes qui y sont développées dépassent le cadre des sciences de la vie et s'adaptent parfaitement aux besoins des sciences sociales.

L'ouvrage aborde conjointement JavaScript et TypeScript : le premier offre une prise en main rapide de la programmation web, tandis que le second apporte la rigueur indispensable à des projets plus ambitieux. **Ensemble, ils constituent le socle idéal pour concevoir des applications data fiables et interactives.**

Ce livre s'adresse aux **étudiants de biologie**, aux **élèves des classes préparatoires BCPST**, mais aussi aux **étudiants en sciences sociales**. C'est un guide précieux pour les **professionnels** désireux d'élargir leur horizon technique.

Laurent Bloch a dirigé des services informatiques clés au sein d'institutions majeures de la recherche et du supérieur (Insee, Ined, Institut Pasteur, Inserm, Université Paris-Dauphine). Auteur de nombreux ouvrages de référence sur la sécurité et les systèmes d'information, il met aujourd'hui cette expertise au service de la pédagogie en proposant une série d'ouvrages dédiée à la bioinformatique qu'il a enseignée au Conservatoire National des Arts et Métiers (CNAM).

ISBN 978-2-7108-1207-4 • 25 €
www.editionstechnip.com

9 782710 812074