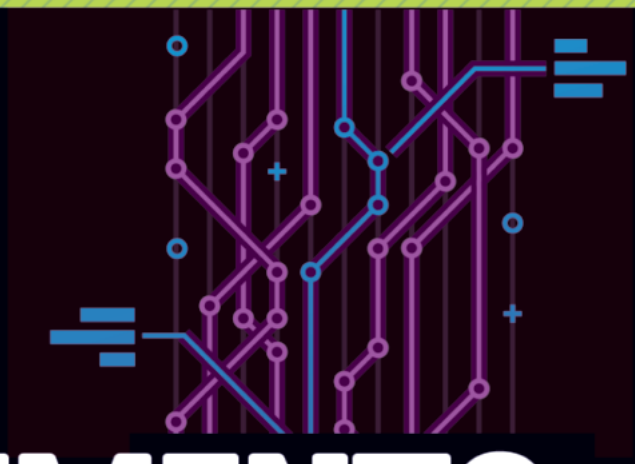
INFORMATIQUE



LES FONDEMENTS DE L'INFORMATIQUE

Du silicium au bitcoin

HUGUES BERSINI
PASCAL FRANCQ
NICOLAS VAN ZEEBROECK

+ EN LIGNE OFFERT

Introduction aux bases de données, au langage SQL, à la programmation orientée objet et à l'informatique d'entreprise

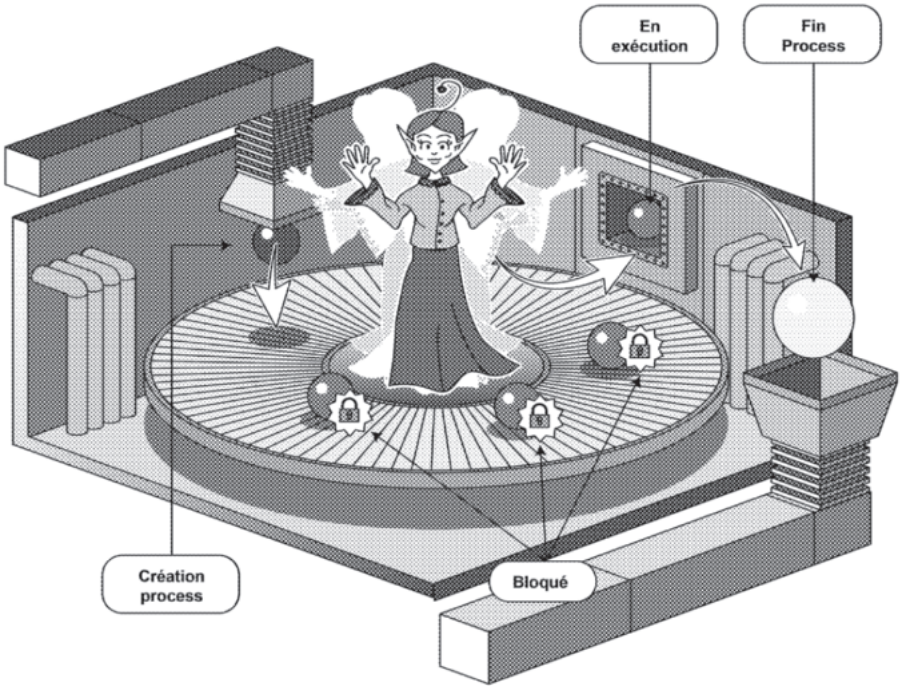
deboeck SUPÉRIEUR

Hugues Bersini,
Pascal Francq,
Nicolas van Zeebroeck

Les fondements de l'informatique

Du silicium au bitcoin

Quatrième édition



Illustrations de Maxime Jacobs

deboeck SUPÉRIEUR

Du même auteur :

H. Bersini, *Algocratie. Allons-nous donner le pouvoir aux algorithmes ?*, préface de Gilles Babinet, 2023

Dans la même collection :

T. Thiry, *Les pratiques de l'équipe agile. Définissez votre propre méthode*, préface de Marc Lainez, 2022
R. Taillet, *Bien débuter en LaTeX*, 2022
B. Lubanovic, *Python. Comprendre les bases et maîtriser la programmation*, 2022
Collectif, traduction de P. Van Goethem & A.-S. Vilret, *Cybersécurité. Sécurisation des systèmes informatiques*, 2021
R. Taillet, *Python pour la physique. Calcul, graphisme, simulation*, 2020
S. Monk, traduction de P. Van Goethem & A.-S. Vilret, *Programmation Arduino. Développez rapidement vos premiers programmes*, 2020
B. Desgraupes, *LaTeX. Apprentissage, guide et référence*, 2019
R. A. Grimes, traduction de P. Van Goethem & A.-S. Vilret, *Hacking et contre-hacking. La sécurité informatique*, 2019

Conception et réalisation de couverture : Primo&Primo
Mise en page : SCM, Toulouse

Dépôt légal :
Bibliothèque royale de Belgique : 2023/13647/095
Bibliothèque nationale, Paris : septembre 2023
ISBN : 978-2-8073-5154-7

Table des matières

Chapitre 5
Unité centrale : processeur et mémoire

Chapitre 6
Périphériques et entrées-sorties

Chapitre 7
Les réseaux

Chapitre 8
Le système d'exploitation

Chapitre 9
Gestion de l'information : du fichier à la Blockchain

Conclusion

Avant-propos

À quoi peut bien servir un livre de culture générale informatique comme celui que vous tenez entre les mains ? À démystifier l'informatique et ce qui l'incarne le mieux aujourd'hui, l'ordinateur, qui est devenu notre compagnon indispensable. Or, le côté paradoxal de cette proximité est que, malgré une présence et une utilisation très familière à la majorité d'entre nous, cette technologie reste mystérieuse à cette même majorité.

Vous nous direz que nous dépendons autant de la voiture, de la machine à café ou de la cuisinière, et qu'il ne semble pas indispensable de s'imposer la lecture d'un ouvrage à leur propos.

Mais alors que la voiture n'est appelée qu'à une seule fonction, l'ordinateur exécute dans la profondeur de son châssis et de sa base matérielle un foisonnement en expansion permanente de logiciels, singularité qui lui confère ce côté multifonctionnel. C'est une machine prête à tout, qui cherche à communiquer avec toutes les autres, et que la plupart des secteurs des sciences, des technologies ou de l'industrie s'empressent de mettre à leur service. Alors, pourquoi ne pas justement mieux la mettre à votre service, en tentant de mieux la comprendre, grâce à ce livre.

Ne nous voilons pas la face : comme vous avez certainement eu l'occasion de vous en rendre compte, l'ordinateur est une machine d'une très grande complexité, peut-être la plus complexe de toutes celles que l'homme a créées. Il réunit en son sein de nombreuses technologies extrêmement variées (électronique, magnétique, optique) qui, associées aux logiciels que ces technologies permettent d'exécuter, aboutissent à une grande sophistication. Les informaticiens tentent de nous masquer autant que possible cette complexité. Toutefois, vous aurez constaté comme nous que, dès le premier problème rencontré, lors d'une installation de logiciels, de périphériques ou de pilotes, d'une mise à jour logicielle ou d'une visite du Web (avec un navigateur puis un autre), cette complexité vous saute au visage et vous laisse souvent décontenancé sinon désarmé. Ce livre devrait contribuer à atténuer ce sentiment, cette envie impérieuse qui vous prend d'ouvrir la fenêtre et de vous débarrasser de votre portable ou autre appareil informatique.

L'informatique génère un vaste vocabulaire technique où termes nouveaux et abréviations absconses ne cessent d'apparaître (le dernier dictionnaire que nous avons consulté contenait presque 5 000 définitions), souvent avec une durée de vie limitée, et tout aussi souvent mal compris par ceux qui en font étalage. Lors de l'acquisition d'un produit informatique, la plupart d'entre nous se trouve confronté à ces mêmes termes dans des brochures publicitaires aussi colorées qu'incompréhensibles. Force est de constater que ce vocabulaire ne fait sens

que pour une très petite minorité d'avertis. Êtes-vous sûr de voir ce qui se dissimule derrière un processeur cadencé à 3 gigahertz, une mémoire cache de 64 kilooctets (et pourquoi se cache-t-elle d'ailleurs ?), une mémoire RAM de 16 gigaoctets ? Et pourquoi doubler la taille de celle-ci est la première opération qui s'impose pour augmenter la vitesse de fonctionnement ? Savez-vous où se niche la carte réseau ou graphique ? Pourquoi les GPU qui composent cette dernière deviennent les processeurs les plus convoités en intelligence artificielle et pour « miner » les Bitcoins ? Avez-vous déjà pris un bus SCSI ou USB ? Connaissez-vous le rôle d'une carte réseau ? En quoi une communication téléphonique et une communication Internet diffèrent ? Et Teams ou WhatApps alors, qu'en est-il ? Comprenez-vous comment fonctionne la compression de données ? Savez-vous expliquer pourquoi des simples documents textes s'échangent parfois mal entre systèmes Windows et Apple ? Et ce qui, à la base, différentie ces deux empires se faisant concurrence depuis 40 ans ? Comprenez-vous les principes sur lesquels reposent la cryptographie, ou le Bitcoin ? Vos données sont-elles à l'abri dans le cloud ?

Savez-vous que, lors de l'achat de votre ordinateur, plusieurs systèmes d'exploitation sont à votre disposition, telles des versions différentes de Windows, GNU/Linux, Unix ou Android ? Que vous pourriez les installer toutes et ce qu'il en coûte d'en choisir une particulière ? Savez-vous pourquoi l'achat d'un jeu ou d'une application logicielle est souvent conditionné par le système d'exploitation ? Que toutes les applications ne fonctionnent pas sur tous les systèmes d'exploitation ? Pourquoi vos périphériques ont-ils besoin d'un pilote ? Pourquoi les opérateurs se disputent-ils les fréquences d'onde radio ?

À toutes ces questions et tant d'autres encore, ce livre apporte un début de réponse. Insistons sur la présence du mot « début ». Il n'est pas question de faire de vous des experts dans les technologies informatiques. Le devenir dans toutes les technologies reprises sous cette appellation est désormais totalement hors de portée, y compris pour la plupart des meilleurs experts. De nombreuses années d'études n'y suffiraient toujours pas, tellement, de la physique quantique aux mathématiques discrètes, de la science des matériaux et des fibres optiques aux technologies télécom, de l'optimisation des circuits électroniques à l'ingénierie logicielle, tant de domaines différents se trouvent impliqués et imbriqués.

Ce livre fait, en revanche, le pari de vous familiariser avec la plupart des technologies auxquelles vous renvoie l'utilisation quotidienne de votre ordinateur et — nouveauté parmi d'autres de cette quatrième édition — le fonctionnement de la Blockchain qui se cache derrière le succès du Bitcoin. Ses dix chapitres, (1) introduction, (2) bases de la programmation, (3) traitement de l'information binaire, (4) codage de l'information binaire, (5) unité centrale : processeur et mémoire, (6) périphériques, (7) réseaux, (8) système d'exploitation, (9) gestion de l'information en ce compris les Bitcoins, (10) conclusions, balaient le large spectre d'utilisation courante d'un ordinateur dans un contexte personnel ou professionnel. Délibérément, nous avons fait le choix d'un exposé concis plutôt que d'une descente en profondeur dans l'une ou l'autre des technologies rencontrées qui, toutes, le mériteraient amplement.

Aux aspects technologiques qui constituent l'essentiel de ce livre, nous avons pris soin, dans cette quatrième édition, d'aborder les principaux enjeux sociaux et environnementaux de l'informatique qu'il n'est plus possible aujourd'hui d'ignorer. Ces enjeux apparaissent désormais de manière transversale au fil des chapitres pour souligner les conséquences environnementales et sociales d'un monde de plus en plus dépendant de l'informatique. Ces fils rouges nous entraîneront parfois sur le terrain de la cybersécurité ou de l'industrie. Ils nous feront voyager dans les nuages, ou descendre dans les mines d'où sont extraits les plus de 70 éléments naturels qui interviennent dans la fabrication de nos microprocesseurs. Chemin faisant, nous découvrirons pourquoi la fabrication de nos équipements est si nuisible à l'environnement, et pourquoi la diffusion de films en très haute définition sur un grand écran constitue l'un des usages les plus énergivores qui soient. Dans le dernier chapitre, nous reviendrons sur ces aspects et sur quelques conséquences directes et indirectes du numérique sur nos institutions socioéconomiques, sur la démocratie et sur la santé humaine.

Le lecteur curieux trouvera également trois compléments accessibles en ligne et par flashcodes à la page 406 du livre :
– une introduction aux bases de données relationnelles et au langage d'interrogation SQL,
– une initiation aux applications et systèmes d'entreprise,
– les fondements de la programmation orientée objet et de la méthodologie de développement logiciel.

Ce livre n'existerait pas sans Marie-Paule et Robert Spinette qui ont été, avec Hugues Bersini, à la base de la première édition de ce livre et se sont encore grandement impliqués dans les deux versions suivantes. Ayant tous deux consacré leur carrière à l'informatique, l'ordinateur n'avait aucun secret pour eux. Robert nous a quittés en 2020, laissant un grand vide derrière lui. C'est tout naturellement que cette quatrième version des fondements de l'informatique leur est dédiée. Nous profitons également de l'occasion pour remercier Maxime Jacobs qui a proposé de repenser avec un véritable talent de graphiste plusieurs des schémas illustratifs.

Bonne lecture, en espérant que, dès la prochaine difficulté rencontrée lors de l'utilisation de votre ordinateur, vous ne vous précipiterez plus chez le vendeur ou sur un téléphone pour vous adresser à quelqu'un qui, dans la plupart des cas, n'en saura guère beaucoup plus que vous…

Chapitre 1

Introduction

N'ayons pas peur des mots, il y va du fonctionnement de l'ordinateur presque comme de celui d'un cerveau : ce dernier procède à l'acquisition de l'information, à sa mise en mémoire, à son traitement et à sa restitution, souvent lors de communications avec ses pairs. Et c'est bien parce que ces deux types de matière grise, l'une en silicium l'autre en carbone, se ressemblent à ce point que le premier se voit de plus en plus substitué, concurrencé ou complémenté par le deuxième, à la grande satisfaction de ceux qui maîtrisent ou contrôlent le premier.

Un scénario pas si futuriste

Bruxelles, un lundi matin, 7 h. Le smartphone d'Emilie entonne Viva la Vida de Coldplay qui lui tient lieu de réveil matin. Quelques minutes plus tard, Alexa a allumé les lumières de l'appartement sur une simple injonction vocale d'Emilie qui sort de sa douche, s'habille et savoure son latte macchiato avant d'ordonner à son assistant vocal d'éteindre les lumières et de verrouiller son appartement. En conduisant sa voiture vers le lieu de sa première réunion du jour, Emilie écoute la DAB sans songer aux dizaines de millions de lignes de code qui s'exécutent pour optimiser le fonctionnement de son moteur, guider son parcours en fonction du trafic, et sécuriser sa conduite. Sitôt installée dans la salle de réunion, elle lance son ordinateur et se branche sur Zoom pour se connecter avec David, son collègue du bureau de Rotterdam. Tout au long de sa vidéoconférence, Emilie a la tête ailleurs. Elle est impatiente de récupérer Tristan, son petit dernier qui doit rentrer ce soir de son voyage en classes vertes. Elle a pu certes suivre ses aventures grâce aux photos envoyées par l'institutrice dans le groupe WhatsApp des parents de la classe, mais elle reste une mère inquiète et ne peut s'empêcher de suivre le voyage du bus scolaire grâce à l'AirTag qu'elle a glissé dans le sac à dos de Tristan.

Perdue dans ses pensées, elle se ressaisit brusquement à la vue du travail graphique de David pour la dernière campagne sur laquelle il travaille : Une photo incroyablement réaliste générée avec l'aide de DALL-E, une intelligence artificielle créée par OpenAI. Une bonne idée de son collègue, qui lui fait songer qu'elle devrait rappeler à son aînée, Clara, de toujours bien vérifier la véracité des élucubrations de ChatGPT et de dûment mentionner son usage dans son élocution sur le panda géant. Avant de quitter son bureau après une longue journée, Emilie utilise son smartphone pour passer quelques commandes sur Amazon en même temps qu'un repas que le livreur d'UberEats devrait apporter pile au moment où elle arrive chez elle, du moins si les calculs de Google Maps sont corrects.

Ces brèves anecdotes illustrent combien l'informatique s'est aujourd'hui insinuée dans les moindres recoins de nos existences. Souvent à notre insu, du code informatique travaille pour nous, régulant nos vies professionnelles (où les processus sont souvent régis par des systèmes d'information) comme privées (oserait-on se rappeler le certificat COVID numérique ?) voire intimes (combien de relations se sont-elles formées grâce à Tinder et autres plateformes de rencontres numériques) ? Or derrière tout logiciel qui s'exécute, un ordinateur calcule et planifie. C'est le cerveau-machine.

Le cerveau machine

De toutes les machines existantes, l'ordinateur est indéniablement celle qui s'assimile le plus au fonctionnement humain (dans ce qu'il a de plus cérébral). Il réalise des traitements d'information à une vitesse qu'aucun cerveau humain ne peut aujourd'hui égaler (et pour cause, une milliseconde est nécessaire pour qu'un neurone communique avec un autre alors que moins d'une nanoseconde suffit (un million de fois moins) pour qu'une donnée circule dans un microprocesseur). *Via* le « cloud » et ses milliards de serveurs, n'importe quel appareil numérique est capable de mémoriser infiniment plus d'informations qu'un cerveau ne le peut. Stockage infini, traitement infiniment rapide, nos pauvres cerveaux ne peuvent que jeter leurs piètres neurones et leurs connexions au pied de la machine.

Par conséquent, sans conteste, l'informatique s'avère la technologie la plus importante du deuxième demi-siècle précédent et du premier demi-siècle actuel, et ce n'est pas par hasard si aujourd'hui nos sociétés se sont vues rebaptisées de « numériques ». Cette nouvelle espèce numérique sera la seule, demain, à pouvoir concurrencer l'homme dans la sélection darwinienne des plus adaptés. Elle commence à voir et à entendre comme lui. Depuis des années déjà, elle se permet de raisonner et calculer comme lui sinon mieux. Elle crée un peu comme lui, de la musique, des images et des textes. Elle commence même à bouger et parler comme lui. C'est vrai que du côté des « émotions », elle reste en retrait. Rien de bien croustillant ne se produit quand un « ordinateur » rencontre une « ordinatrice », mais qu'importe, car comme Platon le clamait, l'émotion pourrait n'être qu'une entrave à nos cogitations. En effet, cette intelligence artificielle n'a de raison d'être que comme expression binaire de la seule raison. C'est bien de QI qu'il doit s'agir ici et de rien d'autre. Et si l'ordinateur tout en raisonnant éprouve quelques chaleurs envahissantes, un ventilateur s'empresse de les dissiper au plus vite (bien que la tendance continue à la miniaturisation rende cela de plus en plus difficile). Finalement, hyper rationnel comme il l'est, l'ordinateur ne croit pas en Dieu, il ne prie pas von Neumann ou Turing, ses créateurs, juste avant de s'éteindre ou juste après s'être allumé… et c'est tellement mieux ainsi.

Notre époque se caractérise par l'omniprésence des ordinateurs. Ils s'insinuent un peu partout. On les trouve en masse dans les chambres des enfants et dans les bureaux des parents, dans les voitures, les machines à laver, les réfrigérateurs, les appareils multimédias ou les aspirateurs. Ils font l'intelligence des robots et des jeux vidéo, et certains informaticiens vont jusqu'à en incorporer dans les vêtements ou dans les aliments. Qui d'entre nous n'est

pas porteur d'une ou plusieurs cartes à puce, qui sont autant d'ordinateurs sous leur forme la plus dépouillée, réduite à ses fonctions essentielles : communiquer, traiter, mémoriser de l'information ?
En effet, stockage, traitement et communication de l'information constituent ainsi les trois invariants technologiques de l'ordinateur. Ce sont ces mêmes invariants qui expliquent son extraordinaire don d'ubiquité et sa faculté unique à se substituer au plus grand nombre d'objets, de fonctions et de machines. Notre ordinateur est la parfaite « machine à tout faire », le « Zelig » fonctionnel incontournable. Écrasé le couteau suisse ! Au grenier la sono, le projecteur de diapositives ou de films, les livres, les disques, les journaux papier, le gigantesque appareil photo d'antan. Tout tient aujourd'hui dans une seule main, sur une branche de lunette et demain sous un ongle ou carrément implémenté dans votre boite crânienne !
L'humanité fit un pas de géant le jour où von Neumann fit le petit pas consistant à faire de cet ordinateur une machine universellement programmable, donc bonne à tout faire, augmentant son pouvoir de substitution et sa polyvalence. Au même titre que l'information à traiter, on peut charger dans la mémoire de l'ordinateur le traitement approprié de cette information, en un mot comme en mille : le « programme ». Très simplement, le traitement devient un deuxième type d'information, à part entière, hormis qu'il manipule le premier. Vous donnez à un ordinateur deux nombres, il les somme, deux textes, il les compare, deux images, il les fusionne, deux musiques, il les superpose…

De l'ubiquité informatique

Un vélo ne fait office que de vélo, une pelle à tarte que de pelle à tarte, mais un ordinateur, lui, peut se transformer en agenda, téléphone, album et appareil photo, banque, radio ou télévision, console de jeu, calculette, système de diagnostic médical automatique, architecte, comptable, bâtisseur, etc. Au cinéma, il devient le second rôle idéal. Dans tous les cas, il est remarquable qu'il s'agisse toujours de la même machine, car il n'y a que le programme à exécuter qui change, excusez du peu… Quand vous chantez ou cuisinez, votre cerveau n'exécute pas le même programme que lorsque vous conduisez, au grand soulagement des automobilistes qui vous croisent sur leur route. Pourtant, ce sont les mêmes neurones qui turbinent. Quand l'ordinateur joue, réfléchit ou agit, le processeur n'exécute pas le même programme, pourtant ce sont, là aussi, les mêmes portes électroniques (ses neurones à lui) qui s'ouvrent et se ferment.
À mesure que le progrès matériel et logiciel augmente la puissance de l'ordinateur tout en réduisant son coût et son encombrement, celui-ci envahit tous les recoins des foyers, tous les interstices des objets usités. Comme Alice ou Gulliver, ses constructeurs le font rapetisser sans arrêt, ce qui lui permet de se loger partout, même dans les endroits les plus étranges, exotiques et au cœur des applications les plus complexes. Les plus récentes révolutions technologiques trouvent leur origine dans l'informatique : automatisation, fabrication d'objets 3D, exploration scientifique, robotique, e-commerce, courrier électronique,

Internet, multimédia… Les hommes les plus riches du monde sont pour la plupart des fondateurs d'entreprises essentiellement numériques (Jeff Bezos d'Amazon, Bill Gates de Microsoft, Larry Ellison d'Oracle, Larry Page de Google, Mark Zuckerberg de Facebook, Elon Musk de Paypal, Tesla, SpaceX et OpenAI…). Ils dirigent les sociétés les plus profitables de l'histoire économique. Le Web est devenu, tout à la fois, la plus grande bibliothèque, vidéothèque, discothèque, encyclopédie, le quotidien le plus lu, alimenté par tout un chacun et non plus uniquement par quelques élus (artiste, encyclopédiste ou journaliste). Jamais la société n'a été autant « participative », le deuxième Web (dit Web 2.0) ayant remis tout à plat par cette fulgurante désintermédiation qu'il favorise (Uber, Airbnb, Blablacar, Facebook, YouTube et compagnie en sont les purs produits). Ces substitutions-là, au grand dam des nostalgiques du vinyle, du papier, des albums de photos jaunies, des salles obscures ou de la presse écrite, constituent dans l'infini terrain de chasse de l'ordinateur, ses proies les plus tangibles. Les hommes n'ont jamais eu autant à voir, à lire et à entendre et si peu de temps pour le faire. C'est la raison pour laquelle les géants du Web se concurrencent pour capter leur attention, les scotcher à l'écran. Chaque Internaute peut y aller de sa plume numérique, et il se trouvera bien quelqu'un, même à l'autre bout de la planète, pour double-cliquer ou « liker » sa prose, y compris par erreur. Les disques durs composant le cloud se saturent à vive allure de nos productions exhibitionnistes.

Mais cette même machine, insatiable, vise des destinées largement plus ambitieuses, en projetant depuis quelques décennies de se substituer à l'intelligence humaine, à la vie ou moins modestement encore, à la nature. N'oublions pas que Turing, un de ses pères (son deuxième étant von Neumann) l'a dénommée dès sa création « machine universelle ». La presse se fait quotidiennement l'écho d'aboutissements spectaculaires, tels que la défaite de Kasparov face à Deep Blue ou celle de Ke Jie face à AlphaGo, Watson, le logiciel vainqueur du célèbre jeu télévisé américain Jeopardy (l'équivalent de notre « Question pour un champion »), la voiture sans chauffeur, la production étonnante d'images par MidJourney, la richesse étourdissante de vraisemblance des conversations de ChatGPT… Il faut se réjouir de la fidélité aujourd'hui atteinte par les simulateurs de vol qui permettent à des pilotes en chambre un début d'autoformation. Il faut s'effrayer de ces drones autonomes qui décident par eux-mêmes d'assassiner ceux qu'ils auront classés comme terroriste potentiel. La même machine peut donc, une fois encore, s'avérer brillante mais ennuyeuse comme Kasparov, terrifiante et meurtrière telle un pilote de chasse, ou stupide mais aussi chou qu'un bébé golden retriever.

Laissons aux philosophes le soin de polémiquer sur ces questions fondamentales, comme ce que ressent réellement Deep Blue devant le désarroi de Kasparov, et le petit chien robot Aibo de Sony quand son maître le caresse ou le sermonne. Il s'agit des sempiternelles questions portant sur la qualité de la restitution et du rapport entre l'original et le modèle, sur la possibilité d'une intelligence réduite à la simple manipulation de symboles, et tant d'une vie que d'une nature ramenées, là encore, à la seule exécution de fonctions logiques ou mathématiques. Intelligence artificielle, vie artificielle, réalité artificielle… Chassez l'artificiel et il revient au galop.

Les quatre fonctions fondamentales

En dépit de leur extrême versatilité, les ordinateurs ne sont en définitive dotées que de quelques capacités fondamentales : acquérir de l'information, traiter de l'information, transmettre de l'information, et la restituer. La première porte le nom barbare de « dématérialisation ». S'il n'est pas sans évoquer quelque scénario de science-fiction, ce terme masque une idée formidablement puissante : créer de tout objet ou phénomène réel une représentation purement informationnelle. Imaginez le contenu du livre que vous tenez dans les mains sans le papier ni l'encre dont il est fait (ou sans votre liseuse si vous avez opté pour sa version numérique), la musique dépourvue de son enveloppe charnelle qu'est le CD ou le vinyle, le film que vous téléchargez sur Netflix, le journal que vous lisez sur votre smartphone… Dans le monde d'aujourd'hui, cela ne demande plus tant d'imagination que cela. Mais songez maintenant que vos empreintes digitales existent elles aussi certainement quelque part en version dématérialisée (par exemple dans la mémoire de votre smartphone), de même que votre argent (ou vos dettes) qui n'est finalement qu'une inscription dans le grand livre numérique de votre banquier ou dans celui, entièrement partagé, de la blockchain. De même, une foule de détails sur votre vie privée, capturée par votre smartphone au fil de vos pérégrinations sur le Web et les réseaux sociaux, sommeille dans les centres de données de Google, Meta et Microsoft. Songez encore que le lèche-vitrine dont vous raffolez le samedi existe aussi en version dématérialisée, sans rues, ni magasins, ni devantures (si si, on appelle cela le commerce électronique et demain le Métavers), que l'octroi d'un crédit hypothécaire ou d'un contrat d'assurance-vie obéissent à des règles codées dans quelques bouts de logiciel. Phénomènes physiques, objets réels comme processus peuvent être ainsi désincarnés et remplacés par une représentation sous forme d'information ou de code logiciel.

Cette irrésistible dématérialisation tend à extraire de tout ce qu'elle touche la seule véritable information nécessaire à son exploitation : la musique sans disque, l'écrit et l'image sans papier, la valeur de l'argent sans pièce ni billet, le film sans pellicule, jusqu'à une hypothétique vie sans biochimie ni sexe, et une intelligence sans neurone ou autre cellule « inutilement biologique ». Que reste-t-il vraiment de notre matière ? La vie ne se réduit-elle qu'à un pur traitement de l'information, aussi sophistiqué puisse-t-il être ?

Chaque fois que de nouvelles interfaces entre les mondes réels et numériques apparaissent, ce sont de nouveaux pans de la société ou de l'économie qui se dématérialisent et se transforment. Car le progrès informatique n'est pas fait que d'ordinateurs plus rapides, il commence par la disponibilité de représentations numériques des phénomènes du réel. Texte et nombres ont été les premiers objets à acquérir une représentation numérique, ce qui a valu à la machine sa dénomination hybride (« ordinateur » ou machine à ordonner en français, « computer » ou machine à calculer en anglais). Le son et l'image ont suivi rapidement. Aujourd'hui, ce sont de nouveaux domaines qui tombent entre les mains de l'ordinateur : identification biologique (empreintes digitales et vocales, ADN, etc.), géolocalisation (coordonnées GPS), monnaie, formes en trois dimensions (impression 3D),

mouvements, transactions et contrats commerciaux, processus industriels, etc. Plus un produit ou un service contient d'information, plus c'est ce contenu en information qui l'emporte sur la matière qui le constitue, et plus il est susceptible d'être dématérialisé à plus ou moins brève échéance. C'est ce que d'aucun philosophe appelle la « silicolonisation du monde ». Et à mesure que de nouveaux types de capteurs apparaissent, ce sont des pans encore plus insoupçonnés et invisibles de nos vies qui se trouvent à leur tour capturés et numérisés.

Si la dématérialisation est un phénomène aussi puissant, c'est parce que l'information est infiniment plus simple à traiter par une machine que des objets réels. Une fois représentée par des 0 et des 1, l'information devient facile à manipuler, analyser, résumer, ou extrapoler. Soudainement réduits à des chiffres binaires, textes et images peuvent soudainement se soumettre à des traitements mathématiques ou statistiques. Le traitement de l'information, en particulier le calcul de prédictions (c'est-à-dire de probabilités) au départ des données, représente la deuxième capacité fondamentale de l'informatique.

C'est ainsi que l'utilisation massive de l'informatique a envahi des disciplines aussi originellement diverses que la chimie, la biologie, la physique, l'économie ou la science du climat. Et les progrès dans tous ces domaines ne seraient rien aujourd'hui sans l'informatique qui leur permet de représenter et simuler les systèmes nerveux, génétiques, immunitaires, économiques, climatiques et bien d'autres. Tous ces sujets d'étude nécessitent pour leur compréhension à la fois le stockage d'une infinité de données et les simulations informatiques qui permettent de leur donner du sens et d'examiner leurs évolutions possibles. Pas de Nobel aujourd'hui sans ordinateur, mais là aussi, l'inverse n'est pas vrai. Les cassandres du réchauffement climatique doivent l'essentiel de leurs prévisions angoissantes à des simulations informatiques prédisant la hausse des températures dans les années à venir. Les systèmes de traduction automatique ont vu leur fonctionnement grandement s'améliorer grâce à la quantité invraisemblable de traductions textuelles existantes et dont les grands modèles linguistiques (« large language models » ou LLM) se servent pour, par simple appariement statistique, réaliser n'importe quelle traduction dans n'importe quel langage que ce soit. La médecine se personnalise de plus en plus grâce au profilage génétique rendu possible par la lecture et le stockage de milliards de bases chimiques constituant notre code génétique et qu'il est possible de regrouper avec d'autres, afin d'affiner diagnostic et pronostic. Ce profilage que chacun rend possible par les traces multiples qu'il laisse sur des myriades de serveurs (achats, commentaires, goûts, mots recherchés, sites Web visités) fit, qu'en 2006, le *Times* élut comme personne de l'année « You ».

Mais tout cela ne serait pas si important sans la troisième capacité informatique, celle qui consiste à transmettre (et recevoir) de l'information à distance, ce que l'on doit bien sûr aux réseaux de télécommunication. Sans la possibilité de dématérialiser l'information, son transfert se heurterait à toutes les limites du monde physique (impossible d'aller plus vite qu'un cheval ou au mieux qu'un avion pour acheminer un message). Mais une fois détaché de son enveloppe physique, un message peut circuler à la vitesse des électrons dans un fil de cuivre ou même à la vitesse de la lumière dans un câble optique. Cette combinaison

« dématérialisation-transmission » a des conséquences redoutables pour de nombreux secteurs d'activité. Tant que l'information n'existait pas en dehors d'un support physique, il suffisait de contrôler la circulation des supports matériels pour contrôler la diffusion de l'information, son intégrité et sa valorisation. Désolidarisée de ce support, l'information peut être, et de manière parfaite, non seulement répliquée à l'infini, mais aussi transférée à des vitesses ultimes aux quatre coins de la planète. Avec l'avènement d'Internet, duplication et transmission de l'information ont perdu leur coût en même temps que toute possibilité de contrôler tous ces transferts.

Paradoxalement, dématérialisation et transmission de l'information ont longtemps mené des vies séparées pour des raisons diverses qui allaient depuis la culture des entreprises concernées jusqu'au codage de l'information (analogique pour le téléphone ou la télévision, numérique pour l'informatique). Aujourd'hui, ce divorce n'est plus qu'un très lointain souvenir et c'est à la confluence de ces deux technologies que nos sociétés et nos pratiques vivent ce qui les bouleverse le plus. Les TIC (ICT en anglais) ou « technologies de l'information et de la communication » sont bien davantage qu'une abréviation éphémère. On assiste à la mutation vertigineuse de nos sociétés en sociétés de l'information et de la communication. L'information se numérise puis se communique. Numérisée, la même information peut se démultiplier à l'infini et se diffuser très largement, soit dans sa version d'origine soit dans une version délibérément pervertie. Espionnages informatiques et virus s'en donnent à cœur joie, d'où l'importance prise comme jamais par les problèmes de sécurité et de confidentialité. Les espions s'invitent partout, les pirates informatiques aussi, et Internet devient un terrain de jeu, d'expérimentation, un champ de bataille et un vivier de victimes potentielles à l'échelle de toute la planète. Quoi que vous fassiez, sur le moindre processeur à votre disposition, cela est enregistré quelque part et susceptible de se retourner contre vous. La cybercriminalité est en passe de devenir la menace la plus sérieuse pour nos démocraties. D'autant que chacun, ravi par la facilité de la chose et contaminé par le virus de l'excroissance narcissique qui envahit nos sociétés, prend un plaisir malin à s'exhiber sans fard aux yeux de millions d'internautes. Rien n'échappe aux webcams qui vous projettent sur YouTube en moins de temps qu'il ne faut pour le dire.

Or, cette convergence de l'informatique et des télécommunications s'accompagne d'une double révolution technologique : celle des vitesses de transfert des réseaux qui repousse les limites des volumes d'information transférables, et celle du sans fil qui confère à l'informatique son caractère mobile et contribue ainsi à son ubiquité. Ainsi ce ne sont pas seulement les objets accessibles à l'ordinateur et les services auxquels il peut se substituer qui s'étendent au rythme de l'innovation technologique, c'est l'espace qu'il conquiert peu à peu.

Pour autant, nous, humains et mortels, restons des êtres de chair et de sang. Nous pourrons certes bientôt converser avec des aliens ou faire l'amour à des créatures chimériques dans le Métavers, mais nos besoins vitaux comme nos expériences sensorielles relèvent bien encore et toujours (et espérons-le à jamais) du domaine réel et matériel. À quoi nous servirait donc de prendre des photos numériques si nos yeux ne pouvaient plus les voir

autrement que comme une séquence mortellement ennuyeuse de 0 et de 1 ? Que vaudrait un abonnement à Spotify ou Netflix si nos appareils ne pouvaient restituer leurs contenus sous forme d'images et de sons ? Heureusement que les informaticiens n'ont pas oublié de concevoir, pour chaque mécanisme de dématérialisation, un mécanisme correspondant de restitution, dotant ainsi l'ordinateur de sa dernière capacité. Il en va ainsi de l'enceinte connectée sur laquelle vous écoutez votre musique préférée, de l'écran qui affiche en arrière-plan la photo du petit dernier, ou encore de l'imprimante 3D qui empile des couches de matière pour constituer l'objet que vous avez dessiné sur votre ordinateur.

C'est au vu de cet état des lieux et des innovations encore à venir — car c'est sans conteste dans cet univers technologique qu'il s'en produit le plus — qu'il nous faut aujourd'hui maîtriser ce monstre virtuel, mieux percer les mystères de ces technologies. Il nous faut comprendre de quoi elles sont faites et comment elles fonctionnent pour en tirer le meilleur parti et comprendre ce que le futur nous réserve. Il nous faut arrêter d'être cet utilisateur béat pendu à son téléphone au moindre problème de démarrage, de chargement ou de mise à jour de logiciel. Il nous faut ouvrir la boîte et décrypter son fonctionnement. Il nous faut comprendre le seul langage que cette machine parle afin de pouvoir communiquer avec elle et la maîtriser.

À l'échelle sociale, nos formes d'organisation et nos interactions sont à réinventer. La technologie est si envahissante, polyvalente et puissante que sa simple intégration dans nos modes de fonctionnement passés ne saurait conduire au meilleur usage possible. Et il suffit de songer à la couverture de l'album de Quick et Flupke intitulé *Vive le progrès !* pour réaliser le chemin qu'il nous reste à parcourir. On y voit Quick (gamin de la ville d'il y a quelques décennies) s'évertuant à battre le tapis du salon suspendu à une corde à linge que sa mère l'a chargé de dépoussiérer. Au lieu de s'y prendre avec une raquette de tennis comme on pourrait s'y attendre, Quick s'épuise à battre violemment le tapis au moyen de l'aspirateur flambant neuf prêté par ses voisins. Merveilleux exemple d'une utilisation de la technologie qui ignore tout de son fonctionnement et de ses potentialités. D'une manière analogue, l'ordinateur ne nous sera réellement des plus utiles que lorsque nous aurons pu imaginer les nouvelles manières de travailler et de communiquer adaptées à son mode de fonctionnement propre, chose impossible si l'on n'en comprend pas les fondements.

Ce livre a pour vocation essentielle de changer cette donne, de vous permettre de regarder l'ordinateur de haut, de convaincre ce partenaire et parfois adversaire de votre infinie supériorité et de lui rappeler sans cesse que si vos cerveaux étaient simples au point de pouvoir être répliqués informatiquement, vous seriez entretemps devenus bien trop bêtes pour y parvenir. Ne le laissez plus vous mettre hors contrôle, hors course, regagnez la maîtrise !

Chapitre 2

L'ordinateur, cette machine programmable

Une histoire abstraite

Autant vous le révéler tout de suite, s'il est bien une chose dont l'ordinateur est totalement dépourvu, c'est d'intention. Un ordinateur ne « veut » ou ne « souhaite » rien, pas plus votre bien que votre mal. Même quand il refuse obstinément de charger sur votre écran le match de football qui oppose les Diables Rouges à l'Équipe de France ou quand il corrige grossièrement et maladroitement votre écriture, l'ordinateur ne vous souhaite rien de mal. Rien de bon non plus d'ailleurs. L'incarnation que nous en proposerons à travers les illustrations de ce livre ne doit pas vous induire en erreur sur ce point : en dépit des prouesses dont il est capable, l'ordinateur n'est qu'une machine. Et la machine ne fait rien, tant qu'on ne lui ordonne pas de faire quelque chose. L'ordinateur a cependant pour lui une qualité unique dans le monde des machines : ses concepteurs ne savent pas à l'avance ce qu'il lui sera demandé de faire. C'est la machine la plus universelle possible, c'est-à-dire qu'elle présente le plus haut degré possible de généralité. En langage plus simple, disons que là où un grille-pain n'est destiné qu'à griller du pain, un lave-linge qu'à laver du linge, une ampoule qu'à éclairer une pièce (et accessoirement à brûler les doigts de celui qui a l'outrecuidance de vouloir la remplacer), l'ordinateur se découvre de nouveaux usages à chaque fois qu'on lui soumet un nouveau programme à exécuter.

Pour commander la machine, il faut et il suffit donc de savoir comment la programmer, c'est-à-dire définir la séquence des opérations qu'elle doit effectuer. Tout traitement de l'information (et donc toute fonction exécutée par un ordinateur) s'opère suivant un algorithme qui peut être défini comme la succession des étapes à réaliser pour accomplir une tâche. Par exemple, la réalisation d'un plat de pâtes fait appel à l'exécution d'un algorithme : verser de l'eau dans la casserole, verser du sel dans l'eau, attendre que l'eau bouille, exécuter une sous-tâche répétitive consistant à prendre des poignées de pâtes dans la boîte et les jeter dans l'eau bouillante. Si vous voulez les pâtes al dente attendez 10 minutes, sinon 15. Égouttez-les et bon appétit. (Un programme mal conçu par un informaticien est souvent désigné de « code spaghetti ».)

Au sujet du mot « algorithme », ne cherchez pas à retrouver sa signification à partir d'une étymologie grecque ou latine, notez le « Al ». Le mot vient de Al'Khawarizmi, ce savant né en Perse qui, au IX^{e} siècle de notre ère, contribua à la diffusion des chiffres et des méthodes de calcul d'origine indienne. Il avait pour principe de « rendre clair l'obscur et simple le

complexe ». Une fois l'algorithme construit, celui-ci est exprimé sous forme d'un programme qui, comme nous le verrons plus loin, est une séquence d'instructions agissant sur les données stockées en mémoire : recherche de ces données, traitement puis réinstallation des résultats dans la mémoire. Ces programmes sont écrits dans un langage dit de programmation, par des programmeurs. Ils les écrivent (la séquence d'instructions, les boucles et les tests conditionnels) comme une suite d'instructions issues d'un langage très précis qu'eux et les membres de leur tribu sont les seuls à comprendre. Malgré cela et un salaire conséquent, ils excellent à rendre ces programmes incompréhensibles aux autres membres de leur tribu. Heureusement, ces programmes seront vérifiés, compris et traduits par un programme particulier (appelé le compilateur) dans un autre langage, fait d'instructions élémentaires qui, elles, seront encore plus incompréhensibles à vos yeux, mais sont les seules comprises et gérables par ce cerveau-machine (le processeur) qui a la responsabilité finale de les exécuter. En définitive, cette exécution se résume à des électrons qui franchissent ou non certaines portes logiques dans une circuiterie invisible à l'œil nu. Mais les utilisateurs finaux, eux, ne se rendront compte (à leurs dépens) si, oui ou non, les programmes se comportent comme souhaité, qu'en observant le résultat à l'écran.

Peut-être cette recension sommaire du fonctionnement général d'un ordinateur vous donne-t-elle l'impression d'un immeuble à plusieurs étages. Ce serait heureux à dire vrai parce que l'analogie est intéressante. À la cave, on trouve la circuiterie électronique, véritable dédale dans lequel des électrons font la course. Au rez-de-chaussée, des portes logiques (ET, OU, et autres conjonctions de coordination bizarres) représentent les circuits électroniques. Montez au premier étage pour faire la découverte des instructions élémentaires, seuls ordres intelligibles pour les étages inférieurs. Le second est l'étage réservé aux programmeurs. On y parle des langages étranges aux noms plus ou moins inspirants : il se dit que des Pythons y font la Java pendant que d'autres C# (prononcez « C-sharp »). L'utilisateur ne se sentira à l'aise (du moins pour les plus chanceux d'entre eux) qu'au troisième, l'étage des applications. C'est ici qu'on clique sur des boutons, ou qu'on pianote sur des claviers pour envoyer des messages à d'autres utilisateurs plus ou moins amis. Leur conversation proprement dite, en réalité, ne prendra son sens qu'au grenier, le seul étage inaccessible à la machine…

Cette allégorie illustre un principe fondamental en informatique qui s'appelle l'abstraction fonctionnelle. Ce principe, qui est le propre de toute l'informatique et même son fondement, dit la chose suivante : l'ordinateur se pense et s'utilise à différents niveaux d'abstraction. Ses tâches, ce qu'il nous importe qu'il fasse, sont programmées de plus en plus haut dans ces niveaux, sans qu'il soit nécessaire de connaître le fonctionnement des niveaux inférieurs, des systèmes de traduction automatique (tels les compilateurs) s'en chargeant pour vous. La raison de ces niveaux tient à l'agrégation des fonctions de bas niveau (les instructions élémentaires) en des fonctions de haut niveau (les clics de souris). On retrouve ici un principe énoncé par Decartes au XVII^e siècle : divisons une tâche en sous-tâches moins complexes, et continuions à subdiviser ces dernières jusqu'à n'obtenir que des problèmes très simples à résoudre.

Comme nous le verrons bientôt, une opération d'addition s'écrit le plus naturellement dans un langage de programmation (par exemple $a = b + c$), mais se décompose en 4 instructions élémentaires pour le processeur (aller chercher *b* en mémoire, aller chercher *c* en mémoire, additionner les deux, stocker le résultat dans *a*). Plus abstrait encore, si vous vous trouvez dans Excel, vous pouvez très simplement, en une opération, calculer la moyenne ou la variance d'une colonne de chiffres. Cette moyenne, réalisée par un clic dans Excel, requiert une succession d'additions suivie d'une division au niveau juste inférieur (programmées dans un langage comme VisualBasic [VB]), chacune à son tour requérant, juste en dessous, quatre instructions élémentaires.

Le tableau ci-dessous illustre ce principe d'abstraction fonctionnelle, où l'utilisateur d'une application peut se limiter à la connaissance des caractéristiques de cette application (par exemple effectuer une moyenne dans Excel), sans avoir à se demander quel langage l'application a été utilisé, quelles instructions ce langage a générées, et enfin quels circuits électroniques de l'ordinateur sont activés par ces instructions.

Définition du problème	élaboration d'un algorithme
Programme d'application	choix par l'utilisateur (programme spécifique, tableur…)
Langage Java	choix par le réalisateur du programme
Assembleur	instructions élémentaires, selon le type de processeur
Logique binaire	résultant de l'assembleur
Circuits électroniques	activés par instructions en binaire

Un autre cas d'abstraction logicielle est représenté par des plateformes qui permettent aux programmeurs d'ignorer le détail de la configuration matérielle et logicielle sur laquelle leurs programmes vont s'exécuter. Un exemple est Microsoft DirectX, qui évite de devoir adapter des programmes tels que les jeux 3D à l'absence ou à la présence d'un des innombrables dispositifs d'accélération matérielle. Le programmeur écrit ce qu'il veut réaliser, et DirectX le transpose en instructions adaptées à la configuration réelle. De manière générale, les progrès dans les technologies informatiques se mesurent notamment à l'aune de cette montée en abstraction qui permet plus d'efficacité pour l'utilisateur à un moindre coût cognitif.

Venons-en aux programmes

Ces fameux programmes sont la seule raison de l'extraordinaire multifonctionnalité propre aux ordinateurs. Ce sont eux qui battent Kasparov aux échecs, font aboyer Aibo (le petit chien robot de Sony), marquent des buts dans un jeu vidéo ou — incorporés dans les robots « Nao » — sur un stade de football, conduisent une voiture (la Google Car par exemple) ou font aux visiteurs d'un site de commerce électronique des recommandations personnalisées

(comme Amazon). Ce sont eux qui permettent les vols d'avions dans leur couloir aérien mais empêchent les vols de produits dans les boutiques, vendent et achètent des valeurs boursières, vous envoient la troisième lettre vous rappelant cette facture en retard de paiement, repèrent et verbalisent vos excès de vitesse, vous disent d'une voix métallique et nasillarde que votre ceinture n'est pas attachée, vous indiquent la meilleure route à suivre et, après tout cela, s'il le faut, vérifient et finissent par assister votre fonctionnement cardiaque.

Au grand dam des programmeurs, les langages de programmation sont multiples, sans doute proches de la centaine, et les convertisseurs automatiques permettant de passer d'un langage à l'autre sont plutôt rares. Les langages de programmation se distinguent entre eux surtout par le niveau de complexité que les fonctions peuvent exprimer, par la concision du langage, et aussi selon le type d'application visé. Les effets de mode, comme partout, sont légion dans ce domaine. Au fil du temps, sont ainsi successivement apparus des langages dits procéduraux (favorisant la découpe du programme en ses grandes fonctions) tel Fortran, plutôt destiné aux applications scientifiques, ou Cobol, plus adapté aux applications administratives et commerciales, et de multiples variantes à vocations plus universelles : Basic, Pascal, C.

Puis sont apparus les langages orientés objet (favorisant cette fois la découpe du programme en ses principaux acteurs, des acteurs interagissant par envois de messages) tels C++, Java, C#, Python, Kotlin, Smalltalk ou Eiffel, et qui ont le vent en poupe ces dernières années. De surcroît, il arrive qu'un même langage de programmation, tels C ou Fortran, puisse présenter des variations dialectales ou syntaxiques dans ses fonctions de base. Ces différences sont introduites par les éditeurs des compilateurs pour des raisons aussi bien techniques que commerciales et sont génératrices d'incompatibilités plus ou moins sévères lors de la compilation du programme ou de son exécution. Des procédures de standardisation sont alors mises en œuvre de manière à ramener chaque langage à une version unique, par exemple la standardisation du C, du Fortran ou du Cobol.

Pourquoi tant de langages de programmation ? Car certains inventeurs ont préféré favoriser l'économie dans la syntaxe de leur langage, d'autres la simplicité, d'autres la lisibilité, d'autres l'optimalité en ressources informatiques, d'autres encore la modularité, etc. Certains inventeurs ont préféré la logique et l'élégance d'une certaine syntaxe à une autre. Certains langages sont la suite logique des précédents, dans lesquels des fonctionnalités nouvelles ont été rajoutées. Ils peuvent alors être incompatibles ou compatibles à des degrés divers avec leurs prédécesseurs. Certains langages n'ont d'autres raisons d'être que commerciales ou se prêter à un type d'applications très précis, comme les langages d'écriture de script pour la réalisation de sites Web actifs et interactifs (PHP, JavaScript) ou pour la réalisation de macros (par exemple VisualBasic, qui vous permet de créer de nouvelles fonctionnalités dans les applications Office de Microsoft). Nous connaissons tant d'informaticiens qui pleurent la belle époque du LISP, du PL1 ou de l'ADA. Et nous en connaîtrons demain qui pleureront la belle époque des Java, C#, PHP, Python, VB ou Delphi.

Il faut bien se rendre compte que l'expansion de l'informatique aurait été totalement impossible sans l'existence des langages de programmation de haut niveau et les progrès considérables accomplis dans ce domaine. Faute de langages évolués, il aurait été impossible de créer la masse actuelle de programmes et de maîtriser leur complexité. Faute de Java, de Python, de C++, pas de Google, de jeux vidéo, de réseaux sociaux ou de ChatGPT. L'immense majorité des informaticiens aujourd'hui passent l'essentiel de leur temps à écrire des programmes ou à adapter des programmes existants, et c'est là souvent l'origine de leurs revenus. Sans ces programmes, il n'y aurait eu que très peu d'intérêt à développer les ordinateurs à l'échelle que nous connaissons. Cet aspect des choses est moins perceptible, car immatériel, mais il n'en reste pas moins un facteur tout aussi important que l'aspect matériel des ordinateurs, que nous pouvons voir et toucher.

Il ne fait pas bon passer le cap de la vingtaine et espérer que vos éventuels talents de programmeur continuent d'être reconnus tant cette pratique informatique est marquée du sceau de la jeunesse. Le « hacker » (traduit en français par « pirate » (alors que tous les hackers ne sont pas des pirates informatiques), « craqueur », et de manière plus péjorative « geek ») type est jeune, très jeune, au look déjanté, l'activation frénétique de ses neurones lui important davantage que son aspect extérieur. Grand consommateur de pizzas et de sucreries, il dépense le plus clair de son temps les yeux rivés devant son écran brillant qui reflète un visage comblé d'aise devant les subtiles et audacieuses lignes de code qu'il pond sans apparente difficulté aucune. Il programme plus vite que son ombre, à la vitesse d'un dactylo expert retranscrivant la lettre urgente que lui dicte sa patronne.

L'informatique est effectivement un domaine technologique dont les entrepreneurs défrayant la chronique n'arborent pas le moindre cheveu gris. Linus Torvalds (concepteur original du noyau Linux), Steve Jobs, Bill Gates, David Filo (un des concepteurs de Yahoo) ou Larry Page (un des concepteurs de Google), n'avaient pas 30 ans quand leurs logiciels envahirent le marché avec le succès fulgurant que l'on sait. C'est souvent à peine sorti de l'université (pour ceux qui en sont sortis : pas Jobs, ni Gates), sans réelle expérience professionnelle, qu'ils choisissent de commercialiser une idée, un algorithme, un programme, qu'ils ont développé au départ en tant que simple projet d'étude. Et c'est bien de programmes qu'il s'agit, comme le fut DOS à l'époque, le système d'exploitation en ligne de commande commercialisé par Bill Gates, ou PageRank, plus récemment, à l'origine du moteur de recherche Google. Ce programme permet d'ordonner les sites Web selon une popularité mesurée par leur position dans le graphe d'hyperliens que constitue le Web (les plus référencés étant les plus populaires). Mark Zuckerberg, concepteur du logiciel de réseaux sociaux « Facebook » est devenu, à moins de trente ans, le plus jeune milliardaire de la planète, et a rapidement rejoint dans le club des plus fortunés ses prédécesseurs de Microsoft, Oracle ou Google.

La programmation est devenue à ce point une discipline clé dans le monde de l'entreprise aujourd'hui, qu'il ne faut guère s'étonner que des entrepreneurs à succès comme Gates et Zuckerberg se font sur le Web les avocats de l'apprentissage au plus tôt de cette pratique (dès les études primaires ou secondaires : voire le site « code.org » où Bill Gates, Zuckerberg

et même l'ancien président Obama participent à cette formation). Voici un petit extrait d'un chapitre de la biographie de Linus Torvalds, qui à 18 ans programmait dans sa chambre le système d'exploitation GNU/Linux, l'antidote gratuit à Microsoft : « *Ce qui rend la programmation si attrayante est que pour pouvoir ordonner à l'ordinateur de faire ce que vous voulez, il faut d'abord savoir comment. Je suis personnellement convaincu que la science informatique a beaucoup en commun avec la physique. Dans les deux cas, il s'agit de comprendre comment fonctionne le monde à un niveau très élémentaire. La différence est bien sûr qu'en physique vous êtes supposé comprendre le monde tel qu'il est construit, alors qu'en informatique, vous créez un monde. Au sein des limites de l'ordinateur, vous êtes le Créateur. Vous avez comme mission ultime de contrôler tout ce qui se passe. Si vous êtes suffisamment bon, vous pouvez devenir Dieu. À une petite échelle…* »

Comment le logiciel a bouleversé l'industrie informatique

Le succès des entreprises logicielles révèle cependant une profonde transformation de l'industrie informatique elle-même. Quand Bill Gates et Paul Allen fondent Microsoft à la fin des années 70, ils sont animés par une vision révolutionnaire pour l'époque : celle que l'informatique de la prochaine décennie ne sera plus gouvernée par les fabricants de matériel mais bien par les éditeurs de logiciels. À cette époque, à peu près contemporaine de la commercialisation des premiers ordinateurs personnels par Apple, l'idée même qu'une entreprise puisse se dédier exclusivement à la production et à la commercialisation de logiciels passe pour saugrenue, tant l'informatique est alors le règne des ingénieurs en électronique qui considèrent le logiciel comme un mal nécessaire pour animer leurs machines. L'industrie informatique est alors organisée autour de quelques grands fabricants (IBM, Digital Equipment, Bull, etc.) qui sont intégrés verticalement et ne servent que des entreprises. Une vingtaine d'entreprises au monde est ainsi responsable de plus de 75 % de toute l'innovation en informatique, et ces entreprises produisent quasiment tous les composants de leurs systèmes elles-mêmes, depuis le processeur jusqu'aux applications.

Mais la vision de Microsoft et surtout l'avènement de l'informatique personnelle va tout changer. À la toute fin des années 1970, IBM — leader mondial de l'informatique d'entreprise — réalise tardivement le potentiel de l'informatique personnelle et combien la machine d'Apple risque de bouleverser l'industrie. IBM se doit de réagir rapidement, et le fait en développant son modèle d'ordinateur personnel (ou PC pour Personal Computer) qui prendra le nom de PS/1 (Personal System 1). Pour gagner du temps dans le développement de cette nouvelle machine, radicalement différente des gros systèmes auxquels Big Blue (c'est le surnom d'IBM) est habituée, elle décide de sous-traiter la fabrication des processeurs à Intel et le développement du système d'exploitation

à Microsoft. Vu leurs tailles respectives à l'époque, IBM ne craint pas une minute que ces deux start-ups puissent un jour lui faire de l'ombre. Mais ni Intel ni Microsoft ne manque de suite dans les idées et tous deux comprennent tout le parti qu'ils peuvent tirer de la situation, à condition qu'ils ne se trouvent pas pieds et poings liés à IBM.

Ils acceptent donc toutes les conditions d'IBM contre une seule clause contractuelle qui leur tient à cœur : la non-exclusivité de leur accord avec IBM, ce qui leur permet de vendre les mêmes solutions (processeurs et systèmes d'exploitation respectivement) à d'autres fabricants. Comme processeur et système d'exploitation définissent ensemble toute l'interface d'une machine et donc ses spécificités, on peut désormais facilement assembler des ordinateurs étiquetés « Compatible IBM ». C'est ainsi qu'est apparue une nouvelle sorte d'acteur dans l'industrie informatique : l'assembleur. De fait, avec la standardisation des processeurs et des systèmes d'exploitation et la disponibilité très large de périphériques, fabriquer un ordinateur personnel s'assimile de plus en plus à un jeu de Lego qui consiste à assembler des blocs de plus en plus standardisés : un boîtier, une carte mère, un processeur, une barrette de mémoire centrale, une carte graphique, un disque dur, un clavier, une souris, un écran, et le tour est joué. Les fabricants d'ordinateurs tout droit sortis de leurs garages sont nés et ils prendront vite le dessus sur les grands fabricants. C'est l'histoire à succès des entreprises Dell et Compaq (qui sera rachetée plus tard par HP). Le duo formé par Intel et Windows s'impose si vite et si bien qu'une dizaine d'années plus tard, on ne parle plus de « Compatibles IBM », mais de puces Intel et d'ordinateurs Windows, qui formeront ensemble ce que l'industrie a appelé le duo « Wintel ». C'est ce même duo qui dictera sa loi à toute l'industrie de l'informatique personnelle sans partage pendant trois décennies, une industrie dont IBM a quasiment disparu.

Ce bouleversement de l'industrie s'est traduit par une totale reconfiguration de la chaîne de valeur. D'un modèle verticalement intégré, l'industrie entière a évolué vers un modèle désintégré où chaque acteur s'est spécialisé dans un composant (matériel ou logiciel) de l'ordinateur : les processeurs pour Intel, les mémoires pour Rambus, les adaptateurs graphiques pour NVIDIA, les systèmes d'exploitation pour Microsoft, etc. Ce changement s'est accompagné d'une révolution plus profonde et moins visible que nous évoquions plus haut : le basculement du centre de gravité de l'informatique du matériel vers le logiciel. Puisque les utilisateurs de l'informatique d'aujourd'hui ne sont plus des informaticiens mais Monsieur tout-le-monde, celui qui contrôle l'industrie est désormais celui qui fournit la partie de l'ordinateur avec laquelle l'utilisateur lambda interagit, c'est-à-dire le logiciel (à commencer par le plus important d'entre eux, le système d'exploitation dont nous reparlerons plus loin).

Une initiation à la programmation avec Python

Ce chapitre a pour mission de familiariser le lecteur à quelques rudiments de programmation dans un langage de haut niveau. Nous ne souhaitons pas faire de vous des experts en la matière. Les quelques pages qui vont suivre n'y suffiraient en rien. Il existe de multiples ouvrages enseignant la programmation à différents niveaux et dans les nombreux langages de programmation qui existent à ce jour. Nous souhaitons simplement vous sensibiliser à la richesse et la puissance de cette pratique, en vous permettant d'appréhender la façon dont on « commande » un ordinateur et de vous guider ainsi à travers les différents chapitres du livre qui plongeront dans les différentes couches de l'abstraction fonctionnelle. Ainsi, ne sera couvert ici que le b-a-ba de la programmation, c'est-à-dire ses briques essentielles : les variables, les typages de base, quelques types composés, les instructions d'affectation et de contrôle, le découpage des programmes par l'utilisation de fonctions et une très succincte introduction à l'orienté-objet que nous mettons en supplément.

Nous avons porté notre choix du langage de programmation sur Python, sans autre raison apparente que sa gratuité, sa facilité d'accès, de téléchargement et de mise en œuvre, la simplicité de sa syntaxe, son côté « open source », et le fait qu'il soit interprété, rendant possible l'affichage immédiat des résultats des instructions sans étape préalable de compilation. Cet aspect des choses facilite énormément la pratique d'essai et erreur, le côté prototypage inhérent à la programmation. Python est un langage ayant recherché dès son origine une grande simplicité d'écriture, tout en conservant tous les mécanismes de programmation objet de haut niveau. Il cherche à soulager au maximum le programmeur de problèmes syntaxiques non essentiels aux fonctionnalités clés du programme. Les informaticiens parlent souvent à son compte d'un excellent langage de prototypage qu'il faut remplacer par un langage plus « solide » tel Java, les langages .Net ou C++, lorsqu'on arrive au terme de l'application. Mais avec la puissance des ordinateurs actuels, cette étape de remplacement est rarement nécessaire, et de nombreuses applications fonctionnant en production sont écrites en Python (parfois seules quelques parties critiques sont redéveloppées dans un langage de programmation plus « proche du matériel »).

Revers de la médaille, beaucoup de critiques du langage utiliseront (c'est d'un classique !) la simplicité de sa syntaxe, l'absence de typage explicite et de compilation, et les retourneront contre le langage (le Python se mordant la queue), en arguant de la difficulté d'aborder des projets complexes dans un idiome aussi élémentaire. Tout langage de programmation est toujours à la recherche d'un graal sans doute introuvable (vu le nombre et la diversité des programmeurs) au carrefour de l'efficience machine, de la simplicité d'usage et d'écriture ainsi que de la robustesse. Il en va des guerres de langages comme de celles des religions, difficile de rationaliser ses préférences. Python est une étape non négligeable dans cette quête céleste. Nous ne nous positionnerons pas dans ce débat. Dernier avantage certain, tout ce qui est nécessaire à l'utilisation du langage et à sa compréhension se trouve réuni sur un site unique : www.python.org. Pour aller plus loin, le lecteur intéressé trouvera en page 406 un lien vers une introduction au développement orienté objet en Python.

Il n'en reste pas moins vrai que, par sa simplicité d'usage, Python, à la différence d'autres langages tels Java ou C++, est un choix s'imposant assez naturellement pour cette modeste initiation à la programmation. Ainsi, de nombreuses universités font ce choix ces jours-ci, rangeant les livres de C++ et même de Java dans les rayonnages poussiéreux de bibliothèques où plus personne ne se rend jamais. Il nous semble que seule la mise en pratique des langages de programmation, c'est-à-dire en programmant effectivement, permet d'assimiler ceux-ci, autant que d'appréhender concrètement le fonctionnement de l'ordinateur. Une lecture attentive des différents bouts de code qui vont suivre devrait suffire à faire comprendre et reproduire le fonctionnement de ceux-ci, mais la véritable maîtrise de Python exige de passer un temps conséquent devant son écran. L'objectif que nous nous sommes fixé ici est simple : comprendre pourquoi et comment les langages de programmation sont le seul véritable mode de communication entre le programmeur et l'ordinateur et comment la pratique de ces langages est pour l'informaticien la seule manière de forcer la machine à se comporter à sa guise à lui.

Les variables

Une variable est un nom qui permet de repérer un emplacement précis de la mémoire centrale, c'est-à-dire son adresse. La manière la plus simple de comprendre une variable est de l'assimiler à une boîte aux lettres qui porterait le nom de cette variable et serait accessible à travers lui. Elle fait donc le lien entre le nom de la variable que l'on manipule dans le programme et une adresse où est physiquement sauvegardée notre donnée. Cette notion, simple en apparence, facilite grandement la réalisation des programmes. Les données manipulées par un programme sont stockées le temps de son exécution en mémoire centrale. Les variables permettent ainsi de manipuler ces données sans avoir à se préoccuper de l'adresse explicite qu'elles occuperont effectivement en mémoire. Pour cela, il suffit de leur choisir un nom. Bien entendu, ceci n'est possible que parce qu'il existe un programme de traduction (l'interpréteur) du programme qui, à l'insu du programmeur mais pour son plus grand confort, s'occupe d'attribuer une adresse à chaque variable et de superviser tous les changements d'adresse qui pourraient s'en suivre.
Par exemple, a=5 signifie « à l'adresse mémoire référencée par "a" (c'est-à-dire dans la boîte aux lettres dénommée a), se trouve la valeur 5 » ; mais nous utilisons plus fréquemment l'abus de langage « la variable a vaut 5 ». Étant donné la liberté autorisée dans le choix du nom des variables et pour de simples raisons mnémotechniques, il est préférable, afin d'également améliorer la lisibilité des programmes, de choisir des noms de variables en fonction de ce qu'elles représentent réellement. Ainsi, préférez par exemple des dénominations telles que « montant », « cote », « nombreDePages » (les langages de programmation sont perdus si vous introduisez des espaces dans le nom d'une variable, d'où cette concaténation fort disgracieuse) ou « prix » à x, y et z.

En Python, la simple déclaration d'une variable ne suffit pas à la créer. Après avoir choisi son nom, il est nécessaire d'affecter une valeur initiale à cette variable. Nous aurons donc par exemple pour les trois variables qui suivent :

```
monChiffre=5
petitePhrase="Quoi de neuf ?"
pi=3.14159
```

Comme le montre cet exemple, différents types d'information (nombres, chaînes de caractères…) peuvent être placés dans une boîte aux lettres, et il est capital de connaître comment la valeur qui s'y trouve a été codée. Cette distinction correspond à la notion de type, fondamentale en informatique car elle débouche, in fine, sur le codage binaire de la variable (par exemple, un entier numérique sur 32 bits et un caractère sur 16 bits) ainsi que sur les usages qui peuvent en être faits (on ne peut pas multiplier un caractère par deux et avoir pour prénom « R2D2 » sauf dans les films de science-fiction). Ainsi, nous dirons qu'une variable recevant une valeur numérique entière est du type « int », une variable recevant un réel (autrement dit un nombre à virgule flottante) sera du type « float » et une variable recevant un ou un ensemble de littéraux sera du type « string ». Il s'agit là des trois types dits simples qui nous occuperont pour l'essentiel dans la suite.
L'instruction d'affectation réalise donc la création et la mémorisation de la variable, l'attribution d'un type à la variable, la création et la mémorisation d'une valeur et finalement l'établissement d'un lien entre le nom de la variable et l'emplacement mémoire. Ainsi, les deux instructions suivantes :

```
maVariable=5
maVariable=2.567
```

permettent de simplement modifier le contenu de la boîte aux lettres maVariable. À la première ligne, on indique qu'à l'adresse maVariable est stockée la valeur 5 et, à la seconde, on remplace la valeur stockée à cet emplacement par 2,567. Après exécution des deux lignes, la valeur sauvegardée en mémoire à l'adresse maVariable est donc 2,567. Notons au passage que le type de maVariable a changé : il est passé d'int à float. Cette spécificité est une facilité essentielle de plus permise par le langage Python. À la différence d'autres langages de programmation plus contraignants, Python est dit « typé dynamiquement » et non statiquement. Cela signifie qu'au cours de l'écriture du code, il n'est plus nécessaire de préciser à l'avance le type des informations exploitées dans le programme, celui-ci devenant « implicite ». Python devine le type par la valeur que nous installons dans la boîte aux lettres — et ce type peut changer au fil des valeurs que la boîte va contenir. Le choix entre typage statique et dynamique est de ces controverses qui ont amusé, distrait et même déchiré la communauté informatique depuis la nuit des temps (c'est-à-dire les années 60 en informatique).

Nous pouvons également prendre le contenu de la boîte aux lettres maVariable et le copier dans la boîte aux lettres taVariable par la simple instruction :

```
taVariable=maVariable
```

Le contenu de maVariable reste inchangé car il s'agit d'un recopiage et non pas d'un transport physique à proprement parler (ici la métaphore de la boîte aux lettres montre ses limites). En fait, il s'agit d'une duplication de l'information d'un lieu de la mémoire vers un autre. Si l'on comprend que la valeur affectée à une variable peut évoluer dans le temps, on comprend tout autant qu'il est possible de réaliser toutes sortes d'opérations sur ces variables, autorisées par leur type. Nous détaillerons pratiquement certaines de ces opérations par la suite. Nous allons voir qu'outre l'abstraction offerte par les variables, le langage nous offre toute une librairie d'utilitaires. Citons d'ores et déjà deux fonctions prédéfinies : type(nomDeLaVariable) qui affiche le type de la variable mise entre parenthèses et s'avère très utile pour vérifier ce qui y a été installé jusqu'ici, et print (nomDeLaVariable) qui affiche à l'écran la valeur stockée à l'adresse désignée par la variable nomDeLaVariable.

Les types simples : int

Le code suivant illustre des opérations réalisées dans la console interactive de Python, sur des variables de type entier (int) (les lignes débutant par « # » sont des commentaires en Python — donc non exécutées — et seront utilisées ici pour éclairer le comportement des instructions). Les « >>> » que vous verrez apparaître par la suite sont produits par l'environnement de développement et sont une « invite » à taper vos instructions Python. Les lignes qui ne débutent pas par ces trois signes sont les réponses de Python à l'exécution de la (ou des) ligne(s) qui précèdent. Les lignes en gras représentent des messages d'erreur. Ce sont les mots doux que nous adresse Python quand il rencontre un problème en exécutant nos instructions.

```
>>> maVariable=3
>>> print(maVariable)
3
>>> print(type(maVariable))
<type 'int'>
>>> taVariable=4
>>> maVariable=taVariable
>>> print(maVariable)
4
>>> maVariable=saVariable

# Une erreur apparaît car "saVariable" n'existe pas

Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
```

```
    maVariable=saVariable
NameError: name 'saVariable' is not defined

>>> maVariable=maVariable+3
>>> print(maVariable)
7
>>> maVariable=maVariable/2
>>> print(maVariable)
3.5 # le type a changé et est devenu réel.
>>> maVariable+3=5

SyntaxError: can't assign to operator

# Un peu de sérieux, il faut respecter un minimum de syntaxe
Python

>>> maVariable+=4 # Raccourci d'écriture
>>> print(maVariable)
7.5
>>> 7//2 # les deux barres au lieu d'une donnent la division
         entière
3
```

En plus de ce que nous avions déjà dit sur l'affectation de variables, il est possible de réaliser des opérations mathématiques sur les valeurs contenues dans les variables. Nous verrons dans les chapitres suivants que les opérations mathématiques font partie des premières instructions élémentaires comprises par tout processeur. Ainsi, maVariable=taVariable+1 signifie que la valeur contenue dans la variable maVariable va prendre celle contenue dans taVariable augmentée de 1 (ici on est plutôt en présence de deux classes d'instruction élémentaires : une addition et un recopiage de valeur d'une variable à l'autre). Nous pouvons aussi constater que l'expression maVariable=maVariable+6 signifie que l'on écrase la valeur de maVariable par sa propre valeur augmentée de 6. Autrement dit, on peut comprendre cette expression en disant : « la nouvelle valeur de maVariable est égale à l'ancienne augmentée de 6 ». On peut d'ailleurs en raccourcir l'écriture : maVariable+=6.
Au rayon des erreurs, on peut d'ores et déjà en répertorier deux. La première consiste à effectuer des opérations sur une variable qui n'a pas été initialisée au préalable. Dans l'exemple, on ne peut en effet assigner la valeur de saVariable à maVariable pour la bonne et simple raison que l'on n'a pas assigné avant cela de valeur à saVariable. La ligne maVariable+3=5, quant à elle, est une simple erreur de syntaxe, montrant que la liberté d'écriture n'est pas totale. Il y a certaines règles de syntaxe et conventions à respecter et nous en découvrirons d'autres par la suite. Pour les langages de programmation de type « compilé », c'est le compilateur qui s'occupe en général de repérer ces erreurs de syntaxe, alors que, dans Python, tout se passe à l'exécution. De nombreux informaticiens jugent cela un peu tard pour s'apercevoir d'une erreur, d'où la préférence qu'ils expriment pour

les autres langages requérant l'étape de compilation (et le typage statique que cette compilation requiert). C'est en cela qu'ils jugent ces concurrents plus robustes : de nombreuses « idioties » sont détectées dès la compilation, vous évitant l'humiliation en public d'un plantage lors de l'exécution de votre programme.

Les types simples : float

Le code suivant illustre des opérations élémentaires sur des réels.

```
>>> taVariable=3.0
>>> maVariable=taVariable/2
>>> print (maVariable)
1.5
>>> print(type(maVariable))
<type 'float'>
>>> print (maVariable)
1.5
>>> print(mavariable)

Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    print(mavariable)
NameError: name 'mavariable' is not defined
```

Côté erreur, on peut rajouter que Python est sensible aux majuscules et minuscules, maVariable et mavariable devenant deux variables différentes. Retenons aussi que 3 est int alors que 3.0 est float.

Les types simples : string

Pour en terminer avec ces types simples, le code suivant illustre des exemples concernant des variables de type string (des chaînes de caractères).

```
>>> phrase1="le framework"
>>> phrase2="python"
>>> phrase1+=phrase2
>>> print(phrase1)
'le framework python'
>>> phrase2*=3
>>> print(phrase2)
'python python python'
>>> print(phrase2[1]) # On saute le caractère blanc du début
p
>>> phrase2+=5
```

```
Traceback (most recent call last):
  File "<pyshell#49>", line 1, in <module>
    phrase2+=5
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> phrase2="phrase1"
>>> print(phrase2)
'phrase1'
```

On constate ici que des opérations sur les littéraux sont possibles : le + en présence de deux strings est réinterprété et permet de concaténer ces chaînes de caractères entre elles, alors que le * sert à répéter un littéral. Remarquons également que la chaîne de caractères se présente comme une collection de caractères indexés ; le premier élément de la chaîne étant repéré par la valeur 0, le second la valeur 1, etc. Dans la plupart des langages de programmation, le premier élément d'une collection est toujours indexé par 0, ce qui entraîne le dernier à se trouver indexé par la longueur de la collection moins un. Commencer un tableau à « un » plutôt qu'à « zéro » est une source d'erreurs classique en programmation, et qui fait beaucoup rire dans les chaumières d'informaticiens. Enfin, remarquons que, nonobstant l'absence de typage explicite, il est interdit de concaténer deux éléments de types différents comme par exemple un string et un nombre naturel. À nouveau, l'erreur se produira à l'exécution vu l'absence de typage statique et d'une étape préalable de compilation. Le compilateur, le chien de garde des programmeurs, aurait pu vous éviter tel affront devant vos pairs. Les programmeurs Python, tout aventuriers qu'ils sont, acceptent cette prise de risque et se satisfont de déférer cette responsabilité à la phase d'exécution. Les goûts et les couleurs…

Une chaîne de caractère est distinguable d'autres types de données par la présence des apostrophes (Python ne fait pas la différence entre simple et double apostrophe). En l'absence de celles-ci, la signification de l'expression peut être toute autre comme en témoigne les deux dernières expressions. Dans le premier cas, c'est la variable phrase1 qui est prise en compte, alors que dans le deuxième, c'est le mot « phrase1 ».

Il est important que les opérations se fassent sur des données de même type comme les trois instructions qui suivent devraient aisément vous le faire comprendre :

```
>>> print("18"+8445)
```

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print("18"+8445)
TypeError: Can't convert 'int' object to str implicitly
```

```
>>> print("18"+str(8445)) # str() force le typage en string
188445
>>> print(int("18")+8445) # int() force le typage en int
8463
```

Les types composites

Nous avons jusqu'ici vu des données de type simple à savoir les int, les float et les strings. En marge de ces types dits simples, se trouvent les types composites. Il s'agit de types qui regroupent en leur sein un ensemble d'entités de type simple. En clair, il s'agit de collections de variables des trois types vus jusqu'ici. Nous en avons déjà vu un cas particulier. En effet, les chaînes de caractères regroupent un ensemble de caractères, chacun de type string. Ici toutes les entités sont du même type. Les autres données composites qui nous intéresseront par la suite sont les listes et les dictionnaires. L'utilité de ces collections est de permettre un même traitement répété sur l'ensemble des éléments composant la collection. Par exemple, si l'on souhaite mettre toutes les lettres d'un mot en majuscule, il serait stupide de réécrire cette opération autant de fois qu'il n'y a de lettres dans le mot. Autant ne l'écrire qu'une fois, et placer cette opération dans une boucle qui balaiera tout le mot. Collections et instructions de boucle font souvent bon ménage comme nous allons le voir. Une des forces de Python est d'incorporer les listes et les dictionnaires comme des types de base du langage. Notons qu'un élément d'une collection (dictionnaire ou liste) peut être lui-même une collection.

Les types composites : les listes

Comme nous l'avons vu dans le cas des chaînes de caractères, chaque élément de la chaîne est indexé par une valeur entière numérotée à partir de 0. Les listes n'en deviennent donc qu'une généralisation à d'autres types simples des chaînes de caractères. Le code suivant clarifie leur utilisation.

```
>>> liste=["journal",9,2.7134,"pi"]
>>> print(liste[0])
'journal'
>>> print(liste[3])
'pi'
>>> print(type(liste[1]))
<type 'int'>
>>> print(liste[4])

Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    print(liste[4])
IndexError: list index out of range

# Erreur car on a dépassé la capacité de la liste !!!

>>> liste.append("bazar")
>>> print(liste)
['journal', 9, 2.7134, 'pi', 'bazar']
>>> liste.insert(4,"truc")
```

```
>>> print(liste)
['journal', 9, 2.7134, 'pi', 'truc', 'bazar']
>>> print(len(liste)) # len() donne la longueur de la liste
6
```

On voit clairement qu'une liste est une simple série d'entités de type simple, chacune étant accessible par l'intermédiaire de son index : 0,1,2… Attention à ne pas dépasser la longueur initiale de la liste, déterminée lors de son initialisation. Pour ajouter un élément en fin de liste, il suffit d'utiliser l'instruction append(), alors que pour insérer un nouvel élément dans la liste en une position précise, il faut utiliser l'instruction insert().
Les indexations des éléments de la liste peuvent également s'effectuer en se référant à l'exemple suivant (sachant qu'une chaîne de caractères est en fait une liste de caractères) :

```
>>> monTexte="hello"+" world"
>>> print(len(monTexte)) # len() donne la longueur de la liste
11
>>> print(monTexte[2])
'l'
>>> print(monTexte[0:2])
'he'
>>> print(monTexte[:4])
'hell'
>>> print(monTexte[6:])
'world'
>>> print(monTexte[-1])
'd'
```

Les types composites : les dictionnaires

Les dictionnaires, autres types composites, sont quant à eux une généralisation des listes. Chaque élément est indexé non plus pas sa position dans la liste, mais par un élément choisi parmi les trois types simples. Il s'agit donc d'une sorte de matrice 2*n d'éléments simples dont ceux de la première colonne réalisent l'index ou la clé (key) d'accès au second.
L'exemple traité dans le code suivant clarifiera les idées :

```
>>> dico={"computer" : "ordinateur"}
>>> dico["mouse"]="souris"
>>> print(dico["mouse"])
'souris'
>>> dico[2]="two"
>>> print(dico)
{2: 'two', 'computer': 'ordinateur', 'mouse': 'souris'}
>>> print(dico.keys())
[2, 'computer', 'mouse']
```

```
>>> del dico[2]
>>> print(dico)
{'computer': 'ordinateur', 'mouse': 'souris'}
```

On peut ainsi remarquer que, pareillement aux listes, les dictionnaires sont un type dynamique. On peut rajouter dynamiquement, c'est-à-dire même après une première initialisation, des éléments au dictionnaire. Ainsi, après la création de celui-ci, nous l'avons enrichi de « souris » et « two » indexé respectivement par « mouse » et 2. Les éléments de la première colonne servent à accéder à leur pendant dans la seconde. L'ordre d'apparition des éléments n'a plus aucune importance dans les dictionnaires car l'indexation de chaque élément ne se fait plus par sa position mais bien par son petit nom stocké dans la première colonne.

Les instructions de contrôle

Jusqu'ici, nous avons vu comment l'interpréteur Python parcourait le code de manière séquentielle. Ainsi, lorsqu'une séquence d'inscriptions est lue, l'interpréteur les exécute les unes à la suite des autres dans l'ordre où elles se trouvent écrites et où il les découvre. Cette exécution systématique présente des limitations dans de nombreux cas. Des instructions de rupture de séquence apparaissent donc nécessaires. Certaines instructions élémentaires des processeurs ont la charge de modifier la séquence naturelle d'exécution du programme en bifurquant vers une nouvelle instruction qui n'est plus la suivante. En substance, il existe deux types d'instruction qui permettent de rompre avec la séquence en cours : la forme conditionnelle et la forme répétitive.

Les instructions conditionnelles

Ce type d'instruction permet au code de suivre différents chemins suivant les circonstances. Il s'agit, en quelque sorte, d'un aiguillage dans le programme. Son fonctionnement est le suivant. Cet aiguillage teste une condition et décide en fonction de la validité ou non de celle-ci, du chemin à suivre dans le programme. La syntaxe est comme suit :

```
if condition:
  bloc1
else:
  bloc2
bloc3
```

Ainsi, le bloc1, un ensemble d'instructions, ne sera exécuté que si la « condition » est vérifiée. Le résultat de l'évaluation de « condition » est donc vrai ou faux — on dit de cette expression qu'elle possède une valeur booléenne. Si le résultat est faux, c'est le bloc2 qui sera exécuté. Dans un cas comme dans l'autre, après son exécution, c'est le bloc3 qui sera entrepris.

Remarquons la présence essentielle des deux points et du retrait. Les deux points marquent la fin de la condition. Tout bloc d'instructions devant s'exécuter, en effet, « comme un seul bloc », doit faire l'objet d'un léger décalage mais indispensable vers la droite. On dit du code décalé vers la droite qu'il est « indenté ». En l'absence de cette indentation, l'interpréteur pourrait donner une erreur, car tout ce qui doit se réaliser, si une condition ou l'autre est vérifiée, doit respecter la même indentation.
L'indentation devient donc fondamentale ici. Elle fait intégralement partie de la syntaxe de Python. Ce qui était une bonne pratique d'écriture de code afin d'en aérer la présentation, généralement suivie par la majorité des programmeurs tous langages confondus, s'est trouvée transformée en une règle d'écriture stricte.
Selon l'indentation choisie pour une instruction quelconque, le résultat à l'exécution pourrait s'avérer très différent, comme l'exemple de code suivant l'illustre quant à la position relative de l'instruction print (« c'est faux »). Cette instruction ne s'exécutera pas dans le premier cas mais bien dans le deuxième, car elle n'est plus conditionnée par la deuxième condition portant sur le « a » (mais elle le reste par la première).

```
>>> a=5
>>> if a<10:
      print("c'est vrai")
    if a>10:
      a=5
      print("c'est faux")

# Résultat à l'exécution du code ci-dessus :

c'est vrai

>>> if a<10:
      print("c'est vrai")
    if a>10:
      a=5
    print("c'est faux")

# Résultat à l'exécution du code ci-dessus :

c'est vrai
c'est faux
```

« else » signifie « autre » en anglais et effectivement le « else » représente tous les cas non couverts par la condition du « if ». On dit aussi du « else » qu'il reprend toute la partie « autrement », car c'est là qu'on aboutit si aucune condition n'aura été validée avant cette partie. Le code suivant clarifiera son utilisation.

```
>>> a=10
>>> if a<10:
        print("a inférieur à 10")
    else:
        print("a supérieur ou égal à 10")

# Résultat à l'exécution du code ci-dessus :

a supérieur ou égal à 10
```

Notez, là encore, la présence des deux points terminant le « else » et l'indentation. Le « else » se trouve comme il se doit en-dessous du « if ».
Imaginons maintenant un aiguillage à plus de deux directions :

```
if condition1:
  bloc1
elif condition2:
  bloc2
else:
  bloc3
```

« elif » est une concaténation de « else » et « if » et pourrait se traduire par « sinon si ». Les conditions doivent être disjointes. En effet, l'ordinateur ne s'accommodant pas d'incertitude, son comportement face à plusieurs conditions validées en même temps serait très ambigu. Dès lors, il faut s'arranger pour que les conditions soient disjointes. Il y a là deux sous-ensembles de situations disjointes, correspondant aux conditions 1 et 2, inclus dans un ensemble englobant correspondant au « else » (et reprenant toutes les autres situations). Remarquons encore que pour un aiguillage à plus de 3 voies, la syntaxe sera :

```
If condition1:
  bloc1
elif condition2:
  bloc2
elif condition3:
  bloc3
else:
  bloc4
```

Un des avantages du « elif » est de pouvoir aligner toutes les conditions disjointes les unes en-dessous des autres, sans être contraint dès lors de décaler de plus en plus vers la droite à chaque condition (la largeur du papier n'y suffirait pas).

Les boucles

Ce type d'instruction permet au programme de répéter, de compter ou d'accumuler tout en permettant une économie d'écriture considérable. La syntaxe de cette instruction est la suivante :

```
while condition:
    bloc
```

Tout d'abord, on se trouve à nouveau en présence des deux points (qui terminent la condition) et du bloc indenté. Tant que la condition est vérifiée, c'est l'ensemble du bloc, c'est-à-dire la suite d'instructions indentées de la même manière, qui va être exécutée. Dès que la condition cesse d'être vérifiée, le bloc est ignoré et l'instruction suivante du code est exécutée. On peut d'ores et déjà voir apparaître deux problèmes : l'initialisation des variables composant la condition avant de procéder à l'évaluation et le risque d'une boucle infinie en présence d'une condition toujours vérifiée. Il faut donc veiller à faire évoluer dans le bloc la valeur d'au moins une des variables intervenant dans la condition et qui permettra à la boucle de s'interrompre.
La boucle est un mode de programmation qui permet d'éviter de réécrire un bloc d'instructions autant de fois qu'on souhaite l'exécuter. Voyez par exemple ceci :

```
compteur=1
while compteur<=2:
  bloc1
  compteur=compteur + 1
bloc2
```

Lors de l'exécution du code qui précède, le bloc1 va être répété deux fois : en effet, au démarrage, le compteur vaut 1. Ainsi, étant plus petit que 2, on va exécuter le bloc1 et incrémenter le compteur de 1. Sa valeur vaut donc 2 à présent. À la fin de l'exécution du bloc, la condition du « while » est à nouveau testée. Cette valeur étant toujours plus petite ou égale à 2, le bloc1 va s'exécuter encore une fois et le compteur passera à 3. À ce stade-ci, la condition n'est plus validée et les instructions subordonnées au « while » seront ignorées. Le bloc2 sera exécuté et le code ira de l'avant.
Évidemment, même si la répétition est ici limitée (seulement 2 fois), on imagine aisément l'économie d'écriture dans le cas où nous souhaiterions exécuter 350 fois le bloc1. Remarquons, pour étayer nos propos, que si le compteur n'était pas incrémenté de 1 à chaque tour, sa valeur restant à 1, la condition serait toujours vérifiée et le bloc1 se verrait exécuter à l'infini, jusqu'à épuisement de l'ordinateur, contraint et obligé de jeter son processeur aux pieds du programmeur. Il faut donc toujours, c'est capital, veiller à ménager une porte de sortie pour toute boucle. Elle vous en saura toujours gré. Le code qui suit illustre la boucle.

```
>>> compteur=0
>>> while compteur<4:
        print(compteur)
        compteur+=1

# Résultat à l'exécution du code ci-dessus :

0
1
2
3
```

L'instruction « for … in … » pour sa part est une autre forme de boucle qui permet cette fois d'itérer sur une collection de données, telle une liste ou un dictionnaire. Le petit exemple ci-dessous l'illustre dans le cas d'une liste. L'avantage est double. Il n'est pas nécessaire de connaître la taille de la collection, et la variable « x » incluse dans le « for in » reprend l'une après l'autre la valeur de tous les éléments de la collection.

```
>>> liste=["Bob","Jean","Pierre","Alain","Yves"]
>>> for x in liste:
        print(x)

# Résultat à l'exécution du code ci-dessus :

Bob
Jean
Pierre
Alain
Yves
```

Dans ce code, la variable « x » incluse dans le « for in » reprend l'une après l'autre la valeur de tous les éléments de la collection. Muni de ces différents éléments, il est aisé de comprendre le code résolvant le petit problème suivant, impliquant un dictionnaire cette fois. Soit une collection de 10 étudiants, chacun avec sa cote obtenue à un examen. Le programme qui suit, et dans lequel de nombreuses lignes ne sont pas montrées, permet d'établir la moyenne à l'examen ainsi que le nombre d'étudiants ayant raté cet examen. Sachez juste que « for x,y in listeEtudiant.items() » permet de boucler à la fois sur la clé et sur les éléments indexés par celle-ci.

```
>>> # On crée un dictionnaire des cotes indexés par les
      étudiants
>>> listeEtudiant["Nicolas"]=16
>>> listeEtudiant["Pascal"]=7
>>> …
>>> listeEtudiant["Hugues"]=10
>>> # On crée les variables pour calculer les statistiques
```

```
>>> Moyenne=0
>>> NombreEtudiants=0
>>> NombreRates=0
>>> for x,y in listeEtudiant.items():
        NombreEtudiants+=1
        Moyenne+=y
        if y<10:
            NombreRates+=1
>>> Moyenne/=NombreEtudiants
>>> print(Moyenne)
10.5
>>> print(NombreRates)
3
```

Autre petit code distrayant, un jeu bien connu des étudiants de l'Université libre de Bruxelles, le jeu du « ding ding bottle ». Le jeu est on ne peut plus simple. Il consiste à énumérer les nombres à tour de rôle et le remplacer par « ding ding » si le nombre est un multiple de 5, et par « bottle » s'il s'agit d'un multiple de 7. Chaque erreur est punie par un gage. Voici dans sa version Python, qui a pour lui d'être d'une précision sans égale, la version de ce petit jeu décapant.

```
>>> while compteur<20:
        if(compteur%5==0) and (compteur%7==0):
            print("ding ding bottle")
        elif(compteur%5==0):
            print("ding ding")
        elif(compteur%7==0):
            print("bottle")
        else:
            print(compteur)
        compteur+=1
```

```
# Résultat à l'exécution du code ci-dessus :

ding ding bottle
1
2
3
4
ding ding
6
bottle
8
9
ding ding
```

```
11
12
13
bottle
ding ding
16
17
18
19
```

Les fonctions

Il est possible dans Python et dans tous les langages de programmation de manière générale de découper le programme en « blocs fonctionnels » appelés « fonction » ou « procédure » et appelables partout dans ce même programme. La programmation de ces fonctions permet à la fois une meilleure modularité du code et s'avère d'autant plus nécessaire que ce même bloc fonctionnel se trouve sollicité à différents endroits du code, évitant ainsi de le réécrire pour chaque appel. C'est l'un des éléments qui rendent l'informatique particulièrement modulaire et donc réutilisable dans différents contextes.
Le comportement et le résultat des fonctions dépendront, comme en mathématique lorsque l'on définit une fonction f(x), d'un ou plusieurs arguments reçus en entrée de la fonction. L'exemple simple illustré ci-dessous est la définition de la fonction « cube(x) » qui renvoie le cube de l'argument x. Après avoir défini la fonction, on l'appelle 4 fois à l'intérieur d'une boucle.

```
>>> def cube(x):
        return x*x*x
>>> compteur=1
>>> while compteur<5:
        print(cube(compteur))
        compteur+=1

# Résultat à l'exécution du code ci-dessus :

1
8
27
64
```

La présence du mot-clé « return » indique la fin de la fonction, en renvoyant le résultat qui suit le « return ». En présence d'un « return », l'exécution de la fonction se termine en renvoyant le contenu du « return » au code appelant. La clause « return » détermine donc la réponse de la fonction à son appel, réponse logiquement dépendante des arguments reçus en entrée au moment de son exécution. Cette réponse de la fonction ne doit surtout pas

LES FONDEMENTS DE L'INFORMATIQUE

Du silicium au bitcoin

Des slogans aux concepts, du vague au concret, du parcellaire au global, telle est la démarche proposée au lecteur dans ce livre qui ne se veut pas une compilation de recettes mais propose plutôt une vision globale de l'informatique, en incluant en particulier les réseaux de télécommunication, qui modifient radicalement sa portée.

Il met d'abord en évidence les concepts sur lesquels repose l'architecture des ordinateurs, puis explique le fonctionnement des éléments majeurs de cette architecture. Il décrit les composants matériels et logiciels communément rencontrés. Enfin, il introduit à la programmation à l'aide du langage Python et à l'interrogation de données avec le langage SQL.

Cette nouvelle édition totalement remaniée prend en compte les nombreux développements intervenus ces dix dernières années : évolution récente des systèmes d'exploitation (Windows, macOS, GNU/Linux), appareils mobiles (Android, iOS), nouvelles interfaces homme-machine (technologies haptiques, commande vocale…), réalité virtuelle et augmentée, Blockchain, IA, Métavers, Cloud Computing, etc…

Hugues Bersini est professeur d'informatique à la Faculté polytechnique et à la Solvay Brussels School of Economics and Management (SBS-EM) de l'Université libre de Bruxelles (ULB). Il est également directeur du laboratoire d'intelligence artificielle de l'ULB et membre de l'Académie Royale de Belgique.

Pascal Francq est ingénieur et docteur en sciences appliquées de l'Université libre de Bruxelles. Enseignant et chercheur, il a fondé le Paul Otlet Institute, un centre de recherche indépendant centré sur les technologies de l'information.

Nicolas van Zeebroeck est professeur à la SBS-EM de l'ULB où il enseigne les systèmes d'information et l'économie numérique. Il est également adjoint des autorités de l'ULB pour l'informatique et le numérique et membre du Conseil Supérieur de l'Emploi en Belgique.

DANS LA MÊME COLLECTION

29,90 €
ISBN : 978-2-8073-5154-7

deboeck SUPÉRIEUR

www.deboecksuperieur.com