Chapitre 1 : Systèmes informatiques

PTSI B Lycée Eiffel

octobre 2020

Le cours d'informatique de cette année se concentrera assez rapidement sur l'étude de deux logiciels bien particuliers, mais avant de rentrer dans des détails plus techniques, nous allons tout de même consacrer quelques heures à un aperçu plus général de ce que représente l'informatique, et de la façon dont cela fonctionne (cette « culture générale » informatique peut très bien en pratique faire l'objet de questions lors de votre évaluation aux concours). Vous vivez aujourd'hui, indiscutablement, dans un monde où l'informatique s'est à la fois révélée complètement indispensable et totalement intégrée dans notre vie de tous les jours. Qui peut imaginer en 2020 se passer d'un téléphone portable ou d'un accès à Internet ? Pourtant, ces inventions sont extrêmement récentes (votre cher professeur y avait à peine accès lors de ses années de lycée). Ce qui fait de l'informatique une science absolument fascinante, c'est que ses développements ont été fulgurants, alors même qu'elle n'est en fait qu'une conséquence assez logique de l'évolution continue de la capacité de l'humanité à communiquer et à transmettre des informations de façon efficace.

1 Une brève histoire de l'informatique.

1.1 Un peu de vocabulaire.

Rappelons simplement pour commencer ce que signifie le terme « informatique ». C'est un néologisme créé il y a quelques décennies (la première utilisation du terme en France date de 1962) à partir des mots « information » et « automatique ». Il désigne donc au départ tout ce qui concerne le traitement automatique de l'information, c'est-à-dire à la fois toute la théorie du traitement de l'information, mais également son aspect pratique, et notamment le fonctionnement des ordinateurs. Le mot « ordinateur » a lui-même été inventé en 1955 par un professeur de lettres à la demande d'un responsable d'IBM. Les deux mots ont d'ailleurs failli faire l'objet d'un dépôt en tant que marques qui aurait empêché leur usage courant, mais ils ont échappé à ce triste sort (notons que ce ne fût par contre pas le cas du mot « informatics » en anglais, d'où l'utilisation outre-Atlantique du beaucoup moins général « computer science » pour désigner en gros tout ce qui est recouvert chez nous par le mot informatique). Pour autant, le traitement automatisé de l'information, ou du moins les premières recherches le concernant, ne datent pas de l'après-guerre.

1.2 Des premiers pas aux premiers ordinateurs.

- vers -3000 : invention du boulier.
- vers 830 : publication des oeuvres d'Al-Khwarizmi.
- 1642 : machine à calculer de Blaise Pascal.
- 1821 : machine à différences de Charles Babbage.
- 1936 : machine de Turing.
- fin des années 1940 : premiers « ordinateurs ».

Le problème principal de l'informatique est vieux comme le monde : la transmission efficace de l'information. De ce point de vue, on peut donc considérer l'invention du langage (nous reviendrons sur ce concept un peu plus loin) comme la première pierre fondatrice de la science informatique. Viendra ensuite celle de l'écriture, qui permettra de communiquer des informations à distance, et de conserver l'information durablement. Depuis cette époque, comme nous le verrons un peu plus loin, les grands principes n'ont guère changé, seuls le support et surtout l'efficacité du système ayant grandement évolué. Bien sûr, ces premières solutions de stockage et de transmission d'information n'avaient rien d'« automatique », cette partie essentielle ne pouvant apparaître que bien plus tard, au gré de progrès autant théoriques que techniques.

Les premiers prémisses de cette automatisation sont apparus en mathématiques, avec la création d'algorithmes de calcul applicables de façon purement mécanique. Rappelons à ce sujet que le mot algorithme est dérivé de celui du mathématicien arabe Al-Khwarizmi, qui décrivit au neuvième siècle des méthodes de résolution d'équations du second degré (dans certains cas particuliers), même si l'algorithme tel que nous le concevons aujourd'hui est une création nettement plus récente (19ème siècle en gros). Il aurait de toute façon été impossible de construire une machine exécutant des algorithmes, même rudimentaires, sans appareillage, mécanique dans un premier temps, puis électronique dans un deuxième, complexe et donc inaccessible avant la révolution industrielle. Si on veut vraiment remonter beaucoup plus loin, Les premiers pas dans le domaine technique sont là encore apparus en mathématiques, comme aides au calcul. On peut remonter quelques millénaires en arrière pour constater l'utilisation du boulier (de façon apparemment indépendante) par plusieurs civilisations différentes, notamment en Chine. Les systèmes mécaniques se complexifieront et se diversifieront dans les siècles suivants, de la règle à calcul à la table de logarithmes (pas une machine à proprement parler). Un premier palier est franchi avec l'invention, en 1642, de la première calculatrice par un certain Blaise Pascal (oui, celui des *Pensées*, qui fut également un brillant mathématicien, notamment à l'origine des premières théories mathématiques des probabilités). Effectuant des additions et des soustractions à l'aide de rouages mécaniques, elle permettait aussi de simplifier les multiplications et les divisions, et sera suivies de plusieurs générations de machines améliorant sensiblement le concept sans le modifier profondément. Un siècle et demi plus tard, en 1801, Jacquard met au point son métier à tisser, commandé par des cartes perforées exécutant un programme, une notion qui rappellera peut-être de bons souvenirs à vos parents (grands-parents ?) s'ils ont connu l'avènement des premiers ordinateurs grand public. En combinant les inventions de Pascal et de Jacquard, le brittanique Charles Babbage conçoit en 1821 une machine analytique qui préfigure l'ordinateur moderne, mais il ne dispose pas d'une technique suffisante pour en achever la construction.

Parallèlement, la théorie continue à avancer à grands pas dans la seconde moitié du 19ème siècle et la première moitié du 20ème siècle (période d'avancées majeures dans tous les domaines scientifiques). Outre la définition claire de ce qu'est un algorithme, le 19ème siècle voit l'apparition de la logique booléenne (logique à deux états, essentielle dans le développements d'outils fondés sur un système binaire). L'application de l'algèbre booléenne aux circuits électriques sera faite en 1938, dans la thèse de Shannon, qui invente accessoirement à cette occasion le terme bit (binary digit) encore utilisé aujourd'hui. L'étape fondamentale suivante sera franchie par Alan Turing, souvent présenté comme le véritable père de l'informatique moderne. À l'origine mathématicien, il s'intéresse au problème de la calculabilité (comment savoir si une proposition donnée est démontrable ou non ?) et crée pour le résoudre son concept de machine de Turing, un modèle abstrait de fonctionnement d'un appareil mécanique de calcul. Tout est désormais en place pour l'avènement de l'ordinateur, dont l'évolution ne sera plus freinée que par les limitations techniques. Quelques inventions du début du vingtième siècle (notamment celle du tube à vide en 1904) permettent toutefois de créer les premiers calculateurs électro-mécaniques dans la première moitié du vingtième siècle. Leur développement est nettement accéléré par la deuxième guerre mondiale, et la nécessité de disposer de machines puissantes (pour l'époque) pour décrypter les messages secrets ennemis. Nous verrons ailleurs dans ce cours que l'informatique doit beaucoup aux militaires, comme d'ailleurs plus généralement la science.

Le premier ordinateur entièrement électronique est généralement considéré comme étant l'ENIAC

(**E**lectronic **N**umerical **I**ntegrator **A**nalyser and **C**omputer), construit à l'Université de Pennsylvanie entre 1944 et 1946, et financé par l'armée américaine. Ce mastodonte contenait pas moins de 17 468 tubes à vide, quelques 70000 résistances et environ 5 millions de soudures (faites à la main !). Il pesait 30 tonnes pour une surface au sol de 167 mètres carrés. Ce n'est en fait pas le calculateur le plus performant de son époque, mais pour vous donner une idée, il permettait d'effectuer la multiplication de deux nombres à 10 chiffres en un millième de seconde (les meilleures machines électromécaniques descendaient difficilement en-dessous de la seconde, et à la main, eh bien, je vous laisse essayer !).Pour la petite histoire, l'ENIAC n'utilisait pas un système binaire de représentation des nombres, mais décimal.

1.3 Les débuts de l'informatique grand public.

- 1965 : loi de Moore.
- 1969 : invention du microprocesseur.
- 1969 : début de l'utilisation d'ARPANET, précurseur d'Internet.
- 1975 : commercialisation de l'Altair 8800.
- 1985 : première version de Windows.

Le développement des ordinateurs ne fera ensuite qu'aller en s'accélérant. L'invention du transistor en 1947, suivie de celle du circuit intégré en 1958, lancent les bases de la miniaturisation des ordinateurs. C'est en 1965 que Gordon Moore conjecture ce qui est connu depuis sous le nom de lois de Moore : une progression exponentielle de la complexité des semiconducteurs, et un doublement du nombre de transistors par puce tous les deux ans (énoncés très souvent repris de façon erronée sous la forme « la puissance doublée tous les 18 mois »). Les lois de Moore ont continué à être approximativement vérifiées jusqu'à la fin du 20ème siècle (la croissance s'est ralentie depuis, mais ça fait déjà un certain nombre de doublements de la capacité !). Moore est l'un des fondateurs (puis président) de la société Intel, qui crée en 1971 le premier microprocesseur (et reste aujourd'hui le numéro 1 mondial de la fabrication de processeurs). Les ordinateurs à taille humaine vont rapidement suivre.

Le premier ordinateur breveté en temps que micro-ordinateur est développé en France en 1973. Mais la première machine conçue pour être vendue à des particuliers est l'Altair 8800 en 1975. Son importance historique est primordiale, dans la mesure où parmi ses acquéreurs figurent Bill Gates, fondateur de Microsoft qui écrira pour cet ordinateur la première version du langage BASIC, ainsi que le duo Steve Jobs et Steve Wozniak, qui créeront de leur côté Apple, dont les premiers microordinateurs enverront très vite aux oubliettes de l'histoire l'Altair 8800 (ils seront notamment les premiers à deviner l'importance d'un périphérique balbutiant à la fin des années 70 : la souris ; l'invention de ce périphérique n'est pas du tout le fait des ingénieurs de chez Apple, mais la commercialisation oui !). Microsoft, de son côté, se concentre sur la production de logiciels, puis celle de systèmes d'exploitation, développant d'abord MS-DOS puis Windows (premier système d'exploitation graphique) au début des années 80. L'ère de l'informatique grand public est réellement lancée.

1.4 L'explosion des technologies et de la commercialisation de l'informatique.

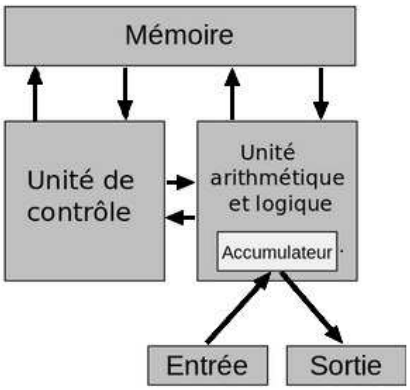
- 1985 : premier ordinateur « portatif ».
- 1990 : première page web.
- 1994 – 1995 : premiers navigateurs web grand public.
- 1996 : première version de Windows CE, précurseur de Windows Mobile.
- 2007 : commercialisation du premier iPhone.

Plus le temps passe, plus la progression de l'informatique et rapide et moins j'ai de commentaires à faire. Cette dernière période est celle dans laquelle nous nous trouvons encore : des ordinateurs de plus en plus puissants, de moins en moins chers, et de plus en plus nombreux. Parallèlement, les nouveautés les plus intéressantes de ces trois dernières décennies sont l'évolution du matériel (les premiers ordinateurs déplaçables ont vu le jour au milieu des années 1980, mais on est surtout passés depuis une quinzaine d'années à d'autres types d'outils technologiques n'étant plus à proprement parler des ordinateurs, comme les smartphone, puis les tablettes, montres connectées et autres consoles de jeu), et la popularité grandissante du travail en réseau permis par Internet. Dernière tendance : la « dématérialisation » de la technologie (normes Wifi ou Bluetooth), qui est bien sûr qu'illusoire. Sans chercher à faire de la science-fiction, la prochaine étape logique sera celle d'un être humain connecté, sans passer par l'intermédiaire d'ordinateurs ou équivalents (là encore il ne s'agirait que d'une illusion, mais une connection via une puce implantée sous la peau et donc « permanente », par exemple, est tout à fait du domaine du réalisable en 2020). Aujourd'hui, un peu plus de 250 millions d'ordinateurs personnels sont vendus chaque année dans le monde (266 millions et quelques en 2019, première année de hausse des ventes de PC après sept années consécutives de baisse, au début des années 2010 les ventes annuelles étaient au-dessus de 350 millions d'unités). Il y a toutefois plusieurs années que les ventes de smartphones ont dépassés celles d'ordinateurs : environ $1,4$ milliard de smartphones vendus en 2019 (un tout petit moins qu'en 2018, après plusieurs années de hausse spectaculaire (on était à moins de 200 millions d'unités vendues en 2009).

2 Architecture matérielle d'un ordinateur.

2.1 Modèle de von Neumann.

Les premiers modèles décrivant la structure générale d'un calculateur (on ne pouvait pas encore parler d'ordinateur à l'époque de leur conception !) sont dus au mathématicien John Von Neumann (plus connu pour sa participation au projet Manhattan, mais qui fût avant tout un très grand théoricien) :



L'**unité arithmétique et logique** (UAL, ALU en anglais) est le coeur de la machine, qui effectue les calculs. La mémoire contient à la fois les données et les programmes à effectuer par la machine (von Neumann n'avait pas anticipé à l'époque la séparation de la mémoire en deux types distincts, telle que nous la connaissons aujourd'hui), et l'unité de contrôle organise les programmes et ordonne les opérations à effectuer. Quant aux entrées-sorties, comme vous l'auriez deviné, elles permettent de communiquer avec l'extérieur. De nos jours, ce modèle a peu évolué, à un détail près : dans l'architecture de von Neumann, les programmes sont stockés dans la mémoire sans faire de différence avec les données, ce qui permet notamment de modifier les instructions du programme en cours d'exécution. On préfère actuellement séparer plus nettement les données des programmes, et ne pas permettre cette modification dynamique des programmes.

Dans les ordinateurs actuels, une unité centrale est en général constituée des éléments suivants :

- la carte mère regroupe est un gros circuit intégré placé au coeur de l'ordinateur, qui regroupe la plupart des composants essentiels au fonctionnement de la machine, notamment :
  — le microprocesseur, « cerveau » de l'ordinateur regroupant à la fois l'UAL et l'unité de contrôle au sens de von Neumann.
  — la mémoire, répartie entre mémoire vive (ou RAM pour **R**andom **A**ccess **M**emory) et ROM (**R**ead **O**nly **M**emory). La ROM stocke des informations « à long terme », qui resteront intactes même lorsque l'ordinateur n'est pas sous tension (tous vos programmes et dossiers de données sont dans la ROM) tandis que la RAM stocke temporairement des informations nécessaires au processeur pendant l'exécution des programmes. Pour cette raison, la RAM est d'accès sensiblement plus rapide que la ROM pour le processeur.
  — le chipset, qui gère les transmissions de données entre les différents composants.
  — les bus internes, qui connectent les différents composants (le terme **bus** est de façon générale utilisé en informatique pour désigner un cablage permettant les transmissions entre différents éléments).
- des bus externes reliant le microprocesseur à des connecteurs permettant eux-même le dialogue avec les périphériques d'entrée-sortie. Ces derniers servent à traduire une information complexe (mouvement de souris par exemple) en langage binaire compréhensible par la machine pour les périphériques d'entrée, ou au contraire traduire le langage binaire en programme exécutable (pour une imprimante par exemple) pour les périphériques de sortie. Ces connecteurs étaient historiquement matérialisés par différents types de **ports** correspondant chacun à un type bien spécifique de périphérique (et capable de traduire un type donné d'informations). Ce système a laissé la place depuis quelques années au principe de l'USB (**U**niversal **S**erial **B**us) qui, comme son nom l'indique, est un modèle de connection permettant de traiter tous types d'informations et donc de s'adapter à tous les types de périphériques. Même ces bus « modernes » sont aujourd'hui à leur tour concurrencés par systèmes de connexion sans fil (WiFi, Bluetooth). Citons pour mémoire quelques types de cablage « historiques » :
  — les ports série et parallèle ont longtemps servi à connecter les principaux périphériques (clavier, souris), avant de laisser leur place au port USB.
  — le connecteur RJ45 (RJ pour Registered Jack) permet la connexion à un réseau (la plupart du temps Internet).
  — les connecteurs VGA, HDMI ou ATA permettent la connexion de périphériques spécifiques comme un écran, un téléviseur ou un disque dur externe.
  — les connecteurs d'extension permettent de brancher un élément améliorant généralement les capacités de l'ordinateur, comme une carte graphique.
- les périphériques proprement dits (qui peuvent être intégrés à l'ordinateur dans le cas d'un portable, ou physiquement indépendants), parmi lesquels on peut citer :
  — pour les périphériques d'entrée, le clavier et la souris bien évidemment, mais aussi la webcam, le micro, ou le lecteur de DVD.
  — pour les périphériques de sortie, l'écran ou l'imprimante.
  — certains périphériques cumulent les fonctions, parmi les périphériques d'entrée-sortie on peut citer le disque dur externe, la clé USB ou même l'écran tactile.

2.2 Logiciels.

Un **logiciel** (en anglais software, par opposition au hardware qui désigne le matériel informatique) est une liste d'instructions (ou programme) donnant accès à l'utilisateur à une interface simplifiée pour effectuer un certain type de tâches sur la machine (on parle de logiciels applicatifs) ou simplifiant simplement la communication avec la machine (logiciels système). L'utilisation de logiciels est absolument indispensable dans la mesure où l'utilisateur est bien entendu incapable de communiquer directement avec le microprocesseur dans le seul langage que celui-ci connaisse, le langage machine.

Elle est d'ailleurs tellement indispensable que la carte-mère comprend elle-même un logiciel, le BIOS (**B**asic **I**nput **O**utput **S**ystem), qui permet d'effectuer quelques opérations élémentaires lors de la mise sous tension de l'ordinateur (et par exemple d'accéder à votre ordinateur le jour où vous avez vraiment salement planté votre Windows). Dans le même ordre d'idée, le système d'exploitation que nous évoquerons au paragraphe suivant est un logiciel sans lequel un ordinateur serait à peu près aussi utile qu'une voiture sans volant.

Les logiciels applicatifs sont extrêmement nombreux et variés. Citons par exemple les logiciels de jeu, les logiciels de bureautique (traitements de textes, tableurs) ou les logiciels de traitement d'image. Les langages de programmation, dont nous parlerons nettement plus intensivement bientôt, sont également des logiciels, même s'ils peuvent eux-même servir à créer d'autres logiciels. Historiquement, la qualité de l'offre logicielle est rapidement devenue un facteur de succès ou d'échec lors de la commercialisation des ordinateurs personnels. Un ordinateur performant sans logiciels pour l'accompagner a autant de chances de percer sur le marché qu'une console de jeux surpuissante, mais qui ne dispose pas d'une offre de jeux de qualité conséquente. Une grande partie de la position dominante de Microsoft sur le marché vient d'ailleurs de la compréhension rapide de ce phénomène : la firme de Bill Gates a réussi à imposer à une vaste majorité d'utilisateurs ses logiciels phares en créant le concept de licence et en imposant la vente du logiciel couplée à celle de la machine. Une nouvelle philosophie s'est développées depuis une quarantaine d'années, celle du logiciel libre, fondée sur les notions de gratuité du logiciel (à l'heure actuelle, la vente de logiciels rapport énormément plus que celle ce matériel informatique) et de partage du code (c'est-à-dire du programme constituant le logiciel), permettant à tous les utilisateurs de modifier eux-même le logiciel. À peu près tous les types de logiciels sont disponibles en version libre (sauf les jeux) à l'heure actuelle, mais leur part de marché reste très minoritaire.

Le **système d'exploitation** (Opérating System ou OS en anglais), tout comme le BIOS décrit au paragraphe précédent (qui en est une sorte de version très rudimentaire), est un logiciel servant d'intermédiaire entre la machine et l'utilisateur. Plus précisément, le système d'exploitation organise toutes les demandes effectuées par les logiciels applicatifs à l'ordinateur : exploitation de la mémoire pour stocker les informations, du processeur pour les calculs etc. Il est en général présenté de nos jours sous forme d'interface graphique cliquable, mais ça n'a pas toujours été le cas, les systèmes d'exploitation plus anciens (MS-DOS notamment) nécessitant de taper au clavier des commandes compréhensibles par la machine. Contrairement au BIOS, et bien qu'il soit souvent vendu avec la machine, le système d'exploitation n'est pas intégré à la carte-mère et il est indépendant de l'architecture matérielle de l'ordinateur. Autrement dit, un PC par exemple n'est pas fait pour tourner sous Windows, et on peut très bien installer un autre système d'exploitation dessus (voire en avoir deux simultanément). Notons au passage qu'en plus du système d'exploitation, les machines sont souvent vendues avec des logiciels d'application (la suite Microsoft Office contenant Word et Excel dans le cas de Windows, par exemple), qui n'ont eux-même rien à voir avec le système d'exploitation, même si un logiciel donné est adapté à un système d'exploitation bien spécifique. Il existe aujourd'hui plusieurs dizaines de systèmes d'exploitation, certains libres, d'autres non (on parle de logiciels propriétaires). Les plus connus sont Windows, Mac OS (pour les propriétaires) et Linux (pour le libre). Il existe également depuis quelques années des systèmes d'exploitation spécialement conçus pour être utilisés sur d'autres types d'appareils numériques que les ordinateurs, comme Android pour les smartphones.

2.3 Réseaux et Internet.

- WWW : **W**orld **W**ide **W**eb
- HTTP(S) : **H**yper**T**ext **T**ransfer **P**rotocol (**S**ecured)
- HTML : **H**yper**T**ext **M**arkup **L**anguage
- URL : **U**niform **R**esource **L**ocator
- TCP-IP : **T**ransmission **C**ontrol **P**rotocol - **I**nternet **P**rotocol

Le principe d'un réseau informatique est fort simple : connecter entre elles un plus ou moins grand nombre de machines pour leur permettre de partager de l'information. Le réseau que vous connaissez le mieux est le plus vaste qui existe aujourd'hui, à savoir Internet, mais on peut aussi très bien créer des réseaux à beaucoup plus petite échelle, par exemple pour permettre à tous les ordinateurs de notre salle de TP de partager des documents. Un réseau comme Internet est constitué de millions de points d'entrée (ordinateurs personnels des utilisateurs, ou serveurs web stockant des pages web) et du cablage nécessaire pour les relier. Sans rentrer dans les détails techniques, cela crée un maillage extrêmement complexe (l'image du filet a donné son nom au web). L'acheminement des données y est donc beaucoup plus complexe qu'au sein d'une machine où le bus représente par exemple l'unique chemin pour aller de la mémoire vive au microprocesseur. Le transport des données sur Internet (on parle de routage) est effectué par paquets (une quantité fixe de données est transférée à chaque étape) et gérée par un ensemble de protocoles connu sous le nom de TCP/IP. Ne comptez pas sur moi pour vous les décrire en détail, le sujet mériterait à lui tout seul un cours complet de plusieurs mois.

3 Représentation des nombres et des données en informatique.

Vous l'aurez désormais compris, le stockage des données est une problématique essentielle en informatique. Cela n'a d'ailleurs rien de spécifique à ce domaine, vous disposez vous-même à l'intérieur de votre crane d'une « machine « (le cerveau) dont l'une des spécificités est de pouvoir stocker de façon efficace de grandes quantités d'information. Dans le cas du cerveau, la façon dont ces informations sont stockées tient encore du mystère, mais pour l'ordinateur, on sait très bien comment ça fonctionne : toutes les données à l'intérieur de votre ordinateur sont ramenées, d'une façon ou d'une autre, à des suites de 0 et de 1, le fameux binaire. Pourquoi ce choix ? Une première nécessité était de ramener tous les types de données possibles (textes, images, programmes) sous une forme commune, pour des raisons évidentes de simplification de la gestion de la mémoire. Ce langage de stockage commun se devait d'être simple et surtout adapté à l'architecture matérielle de l'ordinateur. Celle-ci étant essentiellement constituée, au niveau le plus élémentaire, de dispositifs pouvant prendre deux valeurs facilement identifiables (interrupteur ouvert ou fermé, champ magnétique orienté dans un sens ou dans l'autre), le binaire s'est naturellement imposé (même si, on l'a vu plus haut, ça n'a pas forcément été le premier choix historique).

3.1 Bases et représentation des entiers naturels.

Ce choix du binaire a toutefois un inconvénient : il ne correspond pas à la convention (à peu près) universellement utilisée à l'heure actuelle pour l'écriture des nombres entiers, à savoir le système décimal. En pratique, comment un nombre est-il exactement enregistré dans la mémoire d'un ordinateur ? L'unité de base de mesure de la mémoire est le bit, qui correspond simplement à une unité d'information binaire, autrement dit un 0 ou un 1. Les bits sont ensuite regroupés en octets, qui sont une simple juxtaposition ordonnée de huit bits (les premiers étant traditionnellement appelés bits de poids fort, et les derniers bits de poids faible). Dans un octet, on peut donc stocker $2^8 = 256$ suites de 0 et de 1 différentes, donc 256 informations différentes. On peut donc y stocker

naturellement, par exemple, les valeurs de tous les entiers naturels entre 0 et 255. La représentation binaire du nombre 221 sera ainsi 11011101 si on utilise la conversion classique décimal/binaire.

Rappelons en passant le principe de l'écriture des nombres entiers en **base k**, c'est-à-dire avec une écriture qui utilise k chiffres différents (ainsi, l'écriture décimale usuelle correspond tout bêtement à une écriture en base 10). Quelle que soit la valeur de $k \geqslant 2$, tout nombre entier n peut s'écrire de façon unique sous la forme $n = \sum_{i=0}^{p} a_i k^i$, où chacun des nombres a_i est un entier compris entre 0 et $k-1$ (autrement dit un des chiffres disponibles quand on écrit les nombres en base k). La suite des chiffres $a_p a_{p-1} \dots a_0$ (écrite dans cet ordre) constitue l'écriture de l'entier n en base k. Ainsi, l'écriture binaire de 221 écrite plus haut signifie simplement que $221 = 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^0$ (je vous laisse vérifier que cette égalité est correcte). Convertir un entier binaire en décimal nécessite donc simplement de savoir faire des additions (et de connaitre les premières puissances de 2), c'est facile. La conversion dans l'autre sens est un peu plus délicate, mais peut être effectuée à l'aide de divisions euclidiennes successives par 2 (le même principe permet d'ailleurs de convertir vers n'importe quelle base k, il suffit bien sûr dans ce cas d'effectuer des divisions par k et non plus par 2). On effectue à chaque nouvelle étape le quotient par 2 du précédent quotient obtenu (tant qu'il n'est pas nul) et on garde la suite de tous les restes obtenus. C'est cette suite **écrite en ordre inverse** qui constitue l'écriture binaire du nombre. Par exemple :

221	2							
1	110	2						
	0	55	2					
		1	27	2				
			1	13	2			
				1	6	2		
					0	3	2	
						1	1	2
							1	0

En lisant les restes de bas en haut, on retrouve 11011101.

Remarque 1. La conversion classique de base 10 en base 2 n'est pas la seule façon de coder les entiers de 0 à 255 sur un octet (ou plus généralement les entiers de 0 à $2^n - 1$ sur n bits). Il existe notamment un autre codage parfois utilisé en informatique, connu sous le nom de code Gray (ou code binaire réfléchi). Ce code est assez facile à reconstruire de manière récursive : 0 est toujours codé par le nombre binaire 0, et 1 par la nombre binaire 1. Ensuite, on obtient facilement les nombres compris entre $2^n - 1$ et $2^{n+1} - 1$ en recopiant les $2^n - 1$ premiers nombres déjà obtenus et en ajoutant un 0 devant, puis en recopiant ces mêmes nombres en ordre inverse, en ajoutant un 1 devant. Autrement dit, en codage de Gray sur 2 bits, 0 = 00, 1 = 01 (on a simplement ajouté un 0 devant les deux nombres déjà codés), 2 = 11 et 3 = 10 (on recopie en sens inverse, 1 d'abord et 0 ensuite, et on rajoute un 1 devant). Sur trois bits, on obtient 0 = 000, 1 = 001, 2 = 011, 3 = 010, puis 4 = 110, 5 = 111, 6 = 101 et 7 = 100. Les plus curieux s'amuseront à vérifier qu'on peut obtenir directement le codage de Gray d'un entier décimal quelconque en effectuant le ou exclusif de son codage binaire (standard), avec ce même codage binaire décalé vers la droite (en ajoutant un 0 en premier bit). Quel est l'intérêt d'un tel codage ? Il a la propriété suivante : deux entiers consécutifs ont toujours un code de Gray très proche, plus précisément avec un seul bit de différence (c'est d'ailleurs également vrai pour le passage de $2^n - 1$ à 0).

3.2 Entiers relatifs : le complément à deux.

Les codages présentés au paragraphe précédent ne permettaient de représenter que des nombres entiers naturels en machine. Pour représenter un entier relatif, on aura besoin de deux fois plus de place (en terme de capacité de stockage, ce qui représente en fait un seul bit supplémentaire), puisqu'il faut, en plus de la valeur absolue, préciser son signe. Une solution très simple consiste donc à sacrifier un bit pour indiquer le signe de notre entier, et coder la valeur absolue sur les bits restants.

Ainsi, pour un codage sur un octet, on décide par exemple que le bit de poids fort codera le signe (1 pour un entier négatif, 0 pour un positif), et les sept autres bits la valeur absolue de notre entier. On pourra alors coder sur un octet tous les entiers relatifs compris entre -127 et 127, avec deux codages différents pour l'entier 0. C'est une solution qui n'est pas satisfaisante du tout d'un point de vue pratique, car l'addition binaire de nombres de signes opposés devient très complexe à réaliser.

Il existe en fait une autre possibilité nettement plus satisfaisante, connue sous le nom de complément à 2, qui consiste tout simplement à coder tous les entiers (négatifs comme positifs) modulo 2^n (où n est le nombre de bits utilisés pour le codage), et de considérer qu'on code ainsi (modulo 2^n donc) les entiers compris entre -2^{n-1} et $2^{n-1}-1$. Ainsi, si on code par exemple nos entiers sur 8 bits (en pratique, c'est souvent plus que ça), on va réussir à coder les entiers compris entre -128 et 127 de la façon suivante :

- tout entier naturel inférieur ou égal à 127 garde son code binaire naturel, qui commencera donc (sur 8 bits) par un 0 représentant son signe positif.
- tout entier négatif n supérieur ou égal à -128 sera codé par le code binaire naturel de $n+256$ sur 8 bits. Autrement dit, le code 11011101 qui correspondant dans le paragraphe précédent à l'entier 221 sera désormais utilisé pour coder le nombre négatif $221-256=-35$. Remarquons que tous les codes ayant un premier chiffre égal à 1 représentent donc maintenant des entiers négatifs, le bit de poids fort joue donc le rôle de bit de signe.

Une autre façon de calculer le code de l'entier négatif -35 (qui explique mieux la dénomination de « complément à 2 ») consiste à procéder comme suit : on écrit le code binaire (sur 8 bits) de sa valeur absolue : $35 = 00100011$. Ensuite, on change tous les bits (les 0 deviennent des 1 et vice versa) pour obtenir 11011100. Enfin on ajoute 1 au nombre obtenue (quand je dis « on ajoute 1, c'est-à-dire qu'on fait vraiment l'addition binaire du nombre 1 avec le nombre écrit précédemment, en tenant compte des retenus éventuelles). Ici on retrouve bien $-35 = 11011101$.

Que se passera-t-il dans un programme prévu pour coder les entiers relatifs sur 8 bits si on fait effectuer à la machine un calcul donnant un résultat supérieur à 127 (ou inférieur à -128) ? Tout simplement, le calcul va « boucler » et renvoyer un résultat compris entre -128 et 127, donc la valeur est égale modulo 256 au « vrai » résultat du calcul effectué. Il faut toujours garder à l'esprit que, pour une ordinateur, $\mathbb{N}$, $\mathbb{Z}$ et même $\mathbb{R}$ sont des ensembles finis. Notons quand même que Python est particulièrement bien programmé pour la manipulation des nombres entiers, puisqu'il les code sur un nombre de bits variable qu'il est capable d'adapter lui-même en cas de dépassement. Si vous lui demandez d'afficher la valeur de $3^{3^{10}}$ (nombre qui contient quand même 28 174 chiffres), il le fera de façon complète et exacte.

3.3 Représentation des nombres réels.

Les choses se compliquent encore nettement pour les nombres réels, où il faut coder la partie entière (pas de problème en utilisant ce qui précède), mais aussi ce qui se situe derrière la virgule (et indiquer d'une façon ou d'une autre où se situe cette virgule). Il est par ailleurs évident qu'on ne pourra représenter sur un nombre de bits donné qu'un nombre restreint de nombre réels, et donc imposer à l'avance une certaine précision. La méthode généralement employée consiste en fait à décider dès le départ la précision **relative** qu'on souhaite obtenir, c'est-à-dire le nombre de chiffres significatifs (en binaire, mais le principe serait le même avec des nombres « à virgule » décimaux) qu'on va stocker. Il « suffit » ensuite de préciser où va se situer la virgule par rapport au dernier de ces chiffres significatifs, ainsi que le signe de notre nombre. Imaginons par exemple qu'on cherche à stocker en mémoire le nombre (en écriture décimale) $179,2452$. Ce nombre comporte sept chiffres significatifs, admettons qu'on a assez de place pour tous les stocker (il faudrait en l'occurrence au moins 21 bits pour stocker ces chiffres significatifs en binaire). On va procéder comme suit :

- on stocke d'abord le nombre 1792452 sous forme d'entier naturel (en binaire classique donc), cette partie constitue ce qu'on appelle la **mantisse** de notre nombre réel.

- on stocke à un autre endroit la position de la virgule. Comme ici elle est située « quatre crans à droite » du dernier chiffre significatif, on stockera simplement l'entier -4 en complément à deux, cette partie constitue l'**exposant** de notre nombre réel. Les nombres codés par cette méthode sont souvent appelés nombres flottants, car on parle de « virgule flottante » pour désigner cet emplacement de la virgule décidé par l'exposant.
- enfin on stocke sur un bit le signe : 0 pour notre nombre qui est positif.

Autrement dit, en notant m la mantisse (entier naturel) et e l'exposant (entier relatif) de notre codage, on code ainsi le nombre $\pm m \times 2^e$ (dans l'exemple il faudrait prendre une puissance de 10 et non pas une puissance de 2, bien entendu). Si on décide de coder nos nombres réels sur 64 bits, il est assez traditionnel de réserver 52 bits pour la mantisse et 11 pour l'exposant (le dernier bit étant le bit de signe). On aura donc une précision relative de 52 chiffres en binaires, soit environ 16 chiffres significatifs en décimal. Précions un peu ces histoires de précision relative. Cela ne signifie pas que, par exemple, 2^{-100} est un nombre trop petit pour être représenté en machine. Isolé, il ne présente aucun problème (mantisse bêtement égale à 1 et exposant égal à -100, ce qui tient très très largement sur 11 bits en complément à deux). On pourra de même effectuer un calcul du type $2^{-70} + 2^{-100}$, car l'écart d'ordre de grandeur entre ces deux nombres ne représente que 30 chiffres en binaire, donc moins que la capacité de la mantisse. Par contre, le calcul $2^{-10} + 2^{-100}$ donnera comme résultat 2^{-10}, le deuxième chiffre significatif est beaucoup trop éloigné du premier et sera perdu en cours de route.

3.4 Stockage de chaines de caractères et autres types de données complexes.

- ASCII : **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- ISO : **I**nternational **O**rganization for **S**tandardisation
- UTF-8 : **U**niversal Character Set for **T**ransformation **F**ormat - 8 bits

Savoir représenter des nombres est important, mais évidemment pas suffisant. Tous les autres types de données, on l'a déjà dit, devront être ramenés à un codage numérique, et plus précisément à un codage binaire. Pour terminer cette partie de cours, nous évoquerons simplement le codage des caractères autres que numériques, indispensable au stockage en mémoire de données sous forme de texte. Le principe en est toujours le même (fort simple) : associer à chaque caractère disponible un nombre binaire le représentant (sous forme de table complètement arbitraire de codes associs à chacun des caractères qu'on veut coder, il n'y a plus aucune « logique de calcul » derrière ces codages). Le nombre de caractères codables dépendra donc du nombre de codes disponibles, c'est-à-dire du nombre de bits réservés au codage d'un caractère. Ainsi, si on choisit de coder une liste de caractères sur 8 bits, on pourra représenter 256 caractères différents. L'une des plus anciennes normes de codage de caractères (et d'une grande importance historique) est l'ASCII, qui date du début des années 60, et code sur 7 bits une petite centaine de caractères (certains codages sont réservés à des commandes de contrôle du type « saut de page ») correspondant à peu de choses près aux caractères disponibles sur une machine à écrire américaine classique (pas de caractères accentués notamment). De très nombreuses variantes de ce codage ont ensuite été créées, sur 7 ou 8 bits, permettant notamment d'ajouter les caractères accentués dans les pays les utilisant. Ainsi, en France, la norme de codage la plus utilisée pour le codage des caractères à l'heure actuelle est la norme ISO 8859-15 (parfois également appelée Latin 9 ou Latin 0, c'est une variante de la norme ISO 8859-1 alias Latin 1), qui permet l'affichage des caractères de l'alphabet latin, contenant également les caractères accentués apparaissant dans la plupart des langues européennes (langues nordiques notamment).

Ces normes de codage « locales » (il va de soi que le Latin 9 n'est d'aucune utilité pour un chinois, qui utilisera sa propre table de caractères bien différente de la notre) tendent toutefois à céder du terrain au profit d'une norme de codage internationale regroupant tous les caractères existants dans le monde : le standard Unicode. Les dernières versions de ce standard contiennent plus de 200000

caractères (en particuliers tous les idéogrammes de tous les langues parlées en Chine, par exemple). Ce standard est associée à différentes normes de codage dont la plus répandue est l'utf-8, qui est en particulier compatible avec les autres normes de codage couramment utilisées (si vous configurez votre navigateur web préféré pour un affichage avec un encodage utf-8, il devrait afficher une page web correctement quelle que soit la langue utilisée, ce qui ne sera évidemment pas le cas s'il est configuré pour afficher de l'ISO Latin 9). L'inconvénient de ce type de codage est évident : le très grand nombre de caractères disponible rend le stockage beaucoup plus lourd en termes de place mémoire utilisée (mais l'utf-8 essaye d'être intelligent, en utilisant un nombre de bits variable pour le stockage en fonction de la fréquence d'utilisation du caractère, ce qui complique encore les choses).

Pour finir sur une note beacoup moins sérieuse, la norme de codage ASCII fait l'objet d'une nostalgie suffisante chez certains pour avoir motivé l'apparition d'une discipline « artistique » un peu spéciale : l'ASCII-art, dont le principe est de créer des images en utilisant uniquement les caractères de la table ASCII. Les smileys en mode texte du style :-) en sont des exemples très rudimentaires, mais certains malades ont été jusqu'à recréer des films en images ASCII-art animées (Star Wars en ASCII, le rêve absolu du geek qui sommeille en vous. Non ?).

Les données de types encore plus complexes que du simple texte nécessitent des techniques de stockage encore bien plus sophistiquées. Deux mots sur les images pour conclure : la façon la plus simple de stocker une image est de la stocker pixel par pixel, chaque pixel nécessitant de stocker une couleur. On utilise en général pour représenter une couleur la norme RGB, c'est-à-dire un triplet de trois entiers naturels codés sur 8 bits, représentant les niveaux de rouge, de vert et de bleu de notre couleur. Ainsi, le triplet $[89, 232, 144]$ représente un vert tirant vers le turquoise et assez clair (niveau de vert quasiment maximal donc très lumineux, mélangé à une certaine dose de bleu et moins de rouge). On peut ainsi représenter sur trois octets pas moins de 256^3 couleurs différentes, soit 16 777 216 couleurs (on parle d'ailleurs souvent d'affichage en « 16 millions de couleurs » à ce sujet).

4 Langages de programmation.

4.1 Différents types de langage.

Commençons par une définition très générale : le langage est (je cite Wikipedia !) « la capacité d'exprimer une pensée et de communiquer au moyen d'un système de signes doté d'une sémantique, et le plus souvent d'une syntaxe ». Autrement dit, un langage est un moyent de communication structuré par trois éléments fondamentaux, qu'on retrouve à la fois dans les langages naturels et dans les langages les plus complexes utilisés en informatique :

- un **alphabet** (constitué de symboles ou de phonèmes sonores dans le cas des langages écrits ou parlés par les êtres humains, mais n'importe quel ensemble de signes peut servir d'alphabet, comme par exemple celui du langage des signes).
- une **sémantique** qui donne un sens aux différents assemblages des symboles de l'alphabet (on parlera de mots, comme pour les langages naturels), ainsi qu'aux assemblages de ces mots (c'est la sémantique qui décrète par exemple qu'en français écrit, la suite de symbole « chien » désigne une certaine catégorie d'animaux).
- une **syntaxe** qui est en gros une liste de règles expliquant les constructions qu'on a le droit ou non de créer à partir des mots constituant le langage (c'est plus ou moins ce qu'on appelle aussi grammaire dans le cas du français écrit).

La notion de langage n'a bien sûr rien de scécifique à l'informatique, mais on trouve au sein de celle-ci quantité de langages dont les utilités peuvent être très diverses. Nous nous contenterons ici de donner une petite liste de certaines catégories de langages (parfois de simples exemples) pour illustrer cette diversité :

- les **langages naturels** sont ceux qui sont parlés par les êtres humains, par opposition aux langages formels utilisés en informatique. Celui que vous maîtrisez le mieux (même si très imparfaitement si on en croit Battista) est bien sûr le français (ou plutôt les deux langages, écrit et oral, qu'on désigne par ce nom). Par rapports aux langages artificiels, les langages naturels sont souvent caractérisés par une sémantique très riche (on dispose d'énormément de mots désignant des concepts très proches voire identiques, ce qui ne sera pas forcément le cas dans un langage de programmation, par exemple), et une syntaxe complexe (beaucoup de règles et exceptions peu logiques issues de l'évolution historique du langage) et « floue » : on peut très bien se faire comprendre en français en n'utilisant pas une syntaxe correcte. Heureusement d'ailleurs, mais attention, ce ne sera pas du tout le cas en Python !
- il existe aussi dans notre environnement quantité d'autres langages qu'on pourrait qualifier de naturels même s'ils ne sont pas employés par les humains. Beaucoup d'animaux ont ainsi développé leurs propres façons de communiquer entre eux.
- un **langage informatique** est un langage utilisé lors de l'exploitation d'un système d'information. Il ne s'agit pas nécessairement d'un langage de programmation comme Python (qui servira comme son nom l'indique à écrire spécifiquement des programmes). Ainsi, nous étudierons plus tard dans l'année le langage SQL, qui est un langage de requêtes destiné à effectuer des recherches dans une base de données. Ce cours est tapé dans le langage LaTeX (et mis en page automatiquement par le logiciel sous-jacent) que vous aurez sûrement l'occasion de manipuler également cette année. Un autre langage que vous connaissez tous et qui n'est pas un langage de programmation : le HTML, qui sert à écrire des pages web (pour vous faire une idée de la syntaxe de ce langage, qui utilise énormément le concept de balises, affichez sous votre navigateur web préféré le code source d'une page web pas trop compliquée).
- le **langage machine** est le langage, extrêmement sommaire, que peut directement comprendre un ordinateur. Il s'agit essentiellement d'une suite de 0 et de 1 directement interprétable par la machine. Il est évident que nous n'allons pas apprendre à communiquer avec notre machine directement dans ce langage, et qu'il nous faudra donc un intermédiaire que représentera le langage de programmation (dans le jargon informatique, on dit que le langage machne est le langage de plus « bas niveau » disponible, alors qu'un langage comme Python sera « de haut niveau », c'est-à-dire plus proche d'un langage naturel que du langage machine).
- les **langages de programmation**, qui servent donc à écrire des algorithmes et des programmes capables de les exécuter. À peu près tous les logiciels que vous utilisez quotidiennement ont été programmés dans un de ces langages (il en existe énormément, pour les plus connus, citons l'historique BASIC, le C et son cousin le C++, Java et son compère Javascript, et bien sûr Python). Ces langages n'étant pas lisibles directement par notre ordinateur, ils sont traduits après écriture par un compilateur ou un interpréteur (le compilateur retraduit l'ensemble du programme une fois terminé, alors qu'un interpréteur essaye de comprendre votre programme « à la volée ») qui signalera les éventuelles erreurs de syntaxe (mais pas les erreurs de programmation, si vous écrivez un programme syntaxiquement correct mais qui ne fait pas ce que vous vouliez, ce n'est pas la machine qui peut s'en rendre compte).

4.2 Spécificités du langage Python.

Parmi les centaines de langages de programmation disponibles, il faut faire un choix lorsqu'on décide un beau jour d'apprendre à écrire ses propres programmes. Pour les élèves de classes préparatoires, le choix s'est porté sur Python (et c'est un très bon choix !). Ce langage a été créé en 1990 par Guido van Rossum et fait partie des langages de programmation les plus puissants et les plus utilisés à l'heure actuelle. Il est par exemple utilisé par des entreprises telles que Google (où van Rossum a travaillé quelques années), la NASA ou Industrial Light and Magic (si vous ne connaissez pas, il s'agit d'une petite boîte spécialisée dans les effets spéciaux, qui a notamment créé ceux de *Star Wars*, de *Terminator* ou de *Pirates des Caraïbes* »). Bref, il s'agit d'un « vrai » langage utilisé par des spécialistes, tout en étant (et c'est le plus important pour vous pour l'instant) un langage

adapté d'un point de vue pédagogique par la simplicité et la lisibilité de sa syntaxe (bien sûr, tout est relatif, mais il est assez facile de comprendre le fonctionnement d'un programme Python en le lisant).

Pour rentrer un peu plus dans le détail, sachez que Python est un langage de programmation de haut niveau, à typage dynamique fort, libre, multi-plateformes et multi-paradigmes (orienté objet), doté d'une gestion d'exceptions et d'un ramasse-miettes (garbage collector). Vous n'avez rien compris ? Alors expliquons ces différents termes :

- **de haut niveau** : j'ai déjà expliqué plus haut la distinction entre les langages de bas niveau (proches du langage compréhensible par une machine) et ceux de haut niveau (proches du langage compréhensible par un humain). En particulier, en Python, certaines tâches particulièrement (allocations mémoire par exemple) sont automatisées, ce qui n'est pas le cas de tous les langages de programmation (en C notamment).
- **typage dynamique fort** : nous reviendrons sur cette notion après avoir discuté des variables Python dans le prochain chapitre. L'idée est que Python gère les questions de typage (à quelle catégorie appartient une variable donnée) de manière automatique, contrairement à beaucoup d'autres langages où c'est le programmeur qui doit déclarer le type d'une variable avant même de créer celle-ci.
- **libre** : j'ai déjà évoqué ce terme dans ce cours. Python est un logiciel gratuit (empressez-vous de l'installer sur votre ordinateur personnel), et son code source laissé à libre disposition de tout le monde.
- **multi-plateformes** : le langage est utilisable aussi bien sur un Mac que sur un PC Windows (ou avec d'autres OS plus exotiques).
- **multi-paradigmes** : il existe plusieurs philosophies fondamentales de la programmation, qui consistent à mettre en oeuvre des solutions différentes pour résoudre les problèmes. Sans rentrer dans le détail, les deux principales sont la programmation impérative (la plus classique, qui utilise intensivement le concept de boucles), et la programmation orientée objet, qui consiste à voir toutes les variables d'un programme comme des objets qui vont interagir entre eux (Java fonctionne sur ce principe). Le langage Python est capable de gérer ces deux façons de programmer, même si nous ferons très peu allusion à la programmation orientée objet cette année.
- **gestion d'exceptions** : Python est capable d'interrompre l'exécution d'un programme en cas de problème imprévu.
- **ramasse-miettes** : outil gérant de façon automatique les allocations mémoire et le nettoyage de la mémoire lorsqu'une variable n'est plus utilisée.

Et une dernière anecdote pour la route : pourquoi ce langage s'appelle-t-il Python ? Parce que son créateur était un fan des Monty Python. Si vous voulez vraiment vous mettre dans le bain, filez donc (re)voir *Sacré Graal* !