UN ORDINATEUR

Vous avez tous déjà manipulé des ordinateurs, téléphones mobiles dits *intelligents*... mais savez-vous vraiment comment ils sont structurés et le principe de leur fonctionnement ?

Ce premier chapitre est à caractère essentiellement culturel, cela ne veut pas dire qu'il est dénué d'intérêt pratique. Au contraire.

1 LES ORIGINES

En 1642, Blaise Pascal a l'idée de créer une machine d'arithmétique que l'on appellera plus tard pascaline. Cette machine permettait d'additionner, soustraire, multiplier et diviser aisément des entiers en vue de simplifier le travail comptable de son père qui était super-intendant de Haute-Normandie. Elle permit le premier traitement mécanique d'une information[1].
On y trouve déjà les ingrédients essentiels de l'informatique :

- des données d'entrée : les nombres à manipuler,
- un processus (ici programmé mécaniquement avec les engrenages),
- une sortie : le résultat de l'opération.

En 1837, Charles Babbage invente la première machine à calculer programmable. il s'inspire des métiers à tisser Jacquard[2] dont on pouvait "programmer" le motif du tissage grâce à des cartes perforées.

Vers le milieu des années 1930 apparaissent les premiers ordinateurs. D'abord électro-mécaniques, il s'affranchissent peu à peu des composants mécaniques pour intégrer les lampes à vide puis l'électronique à partir de 1947.

La première erreur informatique est apparue sur un ordinateur à lampe[3]. Un insecte attiré par la chaleur est venu se loger dans les lampes et a créé un court circuit. Il a donné son nom au *bug*.

L'apparition des transistors et des circuits intégrés permet de réduire considérablement la taille des appareils et leur consommation en énergie : on entre dans l'aire des *micro-ordinateurs*. En 1971, le premier micro-ordinateur Kenbach 1 a une mémoire de 256 octets ! Le premier "PC" (*personal computeur*) est commercialisé par IBM en 1981.

Définition 1.1 (Un ordinateur)

Un ordinateur est un dispositif électronique capable de traiter l'information.

Le terme français **ordinateur** a été introduit en 1955 par un responsable du service publicité d'IBM, François Girard. Il s'éloigne de la simple traduction de "computer" (un calculateur) et avance l'idée d'une gestion ordonnée de l'information.

2 COMPOSITION D'UN ORDINATEUR

Un ordinateur est un composant matériel (électronique) doté de logiciels. La partie matérielle est le **Hardware** et la partie logicielle le **Software**.
La structure de base d'un ordinateur (ou de tout autre appareil informatique : smartphone, appareil photo numérique, tablette...) est basée sur la figure 1.

Figure 1: Schéma de base en informatique I/O

[1] L'allemand Wilhelm Schickard avait déjà posé les plans d'une machine semblable quelques années plus tôt, mais ce projet se limita au stade du prototype contrairement à la pascaline qui fut construite en plusieurs exemplaires.
[2] Les premiers métiers Jacquard sont apparus en 1810.
[3] On peut trouver une explication très simple du fonctionnement d'un tube à vide (ou lampe) sur le site http://anjeanje.free.fr/OldHeaven/OldHeavenTubes.htm

Remarque : On pourrait définir une structure un peu plus générale, où l'entrée correspond à un *état* initial, et la sortie à un état final. Le traitement correspond ainsi à un changement d'état et il n'y a donc plus nécessairement de sortie explicite.

A Le Hardware

Pour le Hardware, cette structure peut être résumée par la figure 2.

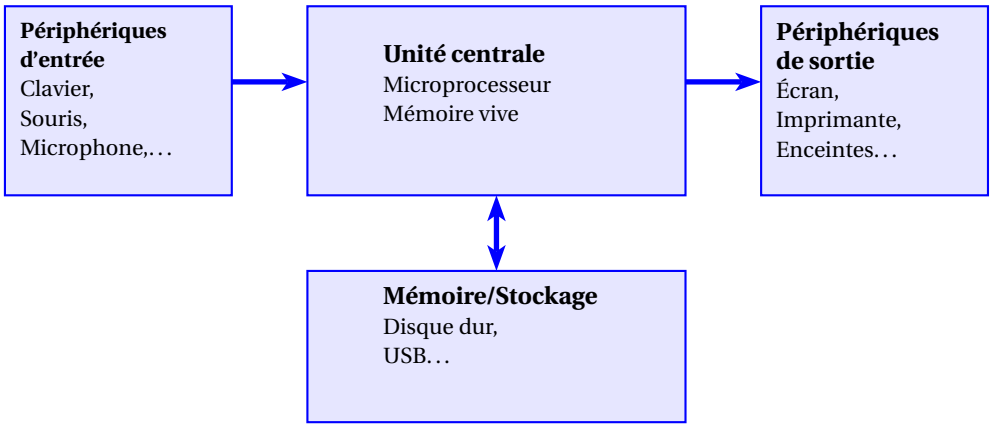


Figure 2: Structure du Hardware

Les **périphériques** d'entrée et de sortie permettent simplement la communication avec l'unité centrale. Cela s'opère via des **unités d'échange** présentes dans l'unité centrale. L'utilisateur transmet ses instructions à l'ordinateur par l'intermédiaire des périphériques d'entrée, et l'ordinateur lui renvoie le résultat par les périphériques de sortie.
La mémoire assure le stockage des données. On distingue la **mémoire vive** (RAM) qui est une mémoire de transfert accessible très rapidement mais de petite capacité (car très coûteuse), de la **mémoire de masse** ou de stockage qui a vocation à conserver les informations sur le long terme.

> Sur les premiers ordinateurs (et jusque vers 1970) les périphériques étaient très rudimentaires et se limitaient à des cartes perforées. En fonction du placement des trous, l'ordinateur comprenait l'information. Il renvoyait ensuite le résultat par carte perforée ou impression. Cette méthode était lourde, fastidieuse et sujette à de très nombreuses erreurs humaines.
> La mémoire n'est pas apparue tout de suite non plus. L'intérêt premier de celle-ci est qu'elle permet de "sauvegarder" des programmes en interne pour des utilisations successives.

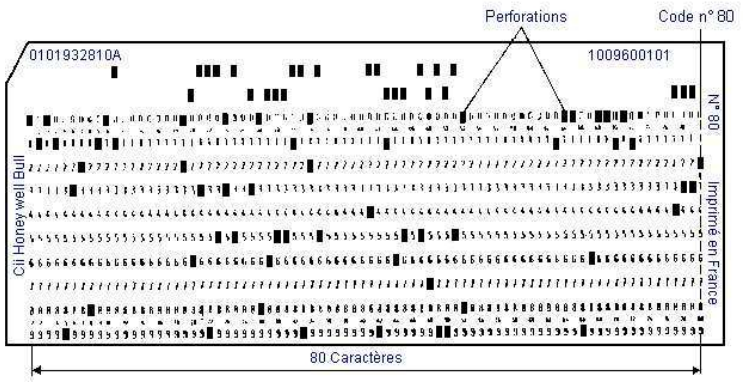


Figure 3: Carte perforée

Le cœur de l'ordinateur est composé de la **carte mère**. Elle comporte certains composants intégrés et d'autres qui lui sont rapportés.

- Le **chipset** est le circuit électronique de la carte qui assure les communications entre les différents organes de l'ordinateur[4]. Le chipset limite les possibilités d'évolution de l'ordinateur en fonction des informations qu'il peut

[4] En général, il contient deux puces, le pont *northbridge* qui gère les communications avec le processeur et les accès mémoire, et le pont *southbridge*

faire transiter (et des connecteurs). Ainsi, il est inutile de brancher un composant trop récent sur une carte mère qui ne le supporte pas car elle ne saura pas l'utiliser.

- Le **CMOS** (*complémentary metal-oxyde semiconductor*) est une mémoire qui contient quelques données essentielles au fonctionnement de l'ordinateur, comme les secteurs disque de démarrage, les configurations matérielles... Cette mémoire est une cible privilégiée des virus car elle reste tout le temps allumée.
  La **pile du CMOS** est une pile plate qui sert à alimenter le CMOS même lorsque l'ordinateur est hors tension. C'est elle qui permet à votre ordinateur d'être à l'heure au démarrage.
  L'**horloge temps réel** (RTC) est chargée de synchroniser les échanges de données. Elle est alimentée par la pile du CMOS.

- Le **BIOS** (*Basic Input/Output System*) est un programme qui communique avec le CMOS (accessible via le BIOS setup) et permet de lancer le système d'exploitation d'un ordinateur. Le BIOS est situé sur la mémoire morte **ROM** (*read-only memory*, originellement, elle ne pouvait pas être modifiée).

- Le **processeur** (ou microprocesseur) exécute les instructions des programmes. Son cadencement définit le nombre d'opérations par seconde qu'il peut exécuter, il est donné par sa fréquence en MHz ou GHz. Un processeur à $1\,GHz$ permet de traiter un milliard d'informations par seconde.

  > L'**overclocking** est une accélération de la vitesse de l'horloge au delà de sa fréquence nominale afin d'augmenter les performances. Mais c'est une technique qui n'est pas sans risque (surchauffe du processeur, erreurs de calcul, instabilité ou destruction de l'OS...)

  La tendance est à la multiplication des processeurs, ou au multi-cœurs. Les architectures parallèles permettent ainsi d'effectuer plusieurs tâches en parallèle et non à la suite les unes des autres.
  Le processeur ne fait pas partie de la carte mère, mais il se branche dessus via le connecteur **SOCKET**. En fonction des gammes de processeurs, les **SOCKET** ne seront pas les mêmes. Le processeur est un élément coûteux de l'ordinateur.

- La **mémoire vive** ou **RAM** (*Random access memory*) est une mémoire d'accès très rapide pour conserver les informations d'usage immédiat. La RAM est entièrement vidée lorsque l'ordinateur est mis hors tension.

  > Si un ordinateur ne possède pas suffisamment de RAM, il utilisera la mémoire de masse (disque dur) pour conserver des résultats intermédiaires. Il fera donc beaucoup de lectures/écritures sur le disque dur qui est beaucoup plus long d'accès que la RAM (environ 100 000 fois plus long). Cela ralentira considérablement les processus.

- Un **dissipateur thermique** est associé à la carte mère pour dissiper la chaleur (produite par effet Joule) et éviter de faire fondre les composants.

  > Les data centers, calculateurs, ou simplement les salles équipées de nombreux ordinateurs sont climatisées pour contrer l'apport thermique dû à l'effet Joule. Les consommations électriques des grands calculateurs et des data centers sont en effet très importantes et représentent un gros enjeu actuel. La consommation des data centers est actuellement estimée à 1% ou 2% de la consommation électrique mondiale.

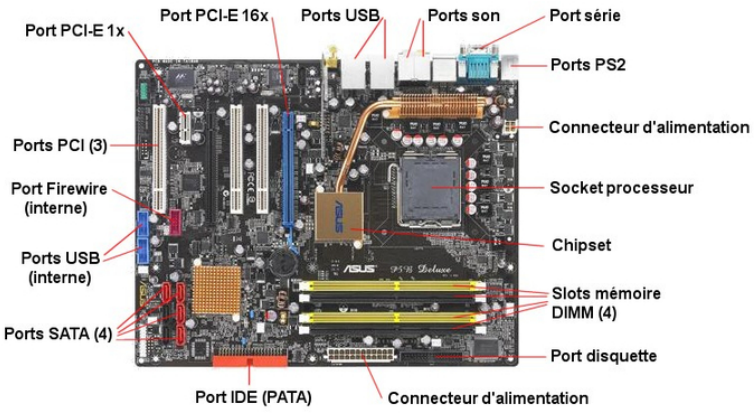


Figure 4: Description d'une carte mère (*source : choixpc.com*)

qui communique avec les périphériques (entrées/sorties). Aujourd'hui, le chipset intègre souvent une puce graphique et une puce audio qui peuvent remplacer la carte graphique et la carte son (mais de qualité moindre).

La **mémoire de masse** correspond globalement au disque dur. La majorité des disques durs sont composés de disques recouverts d'une couche magnétique dont le champ magnétique local code des données (0 ou 1). La mémoire flash est plus récente, elle est utilisé sur les clés USB. Basée sur la technologie des semi-conducteurs, elle permet un accès plus rapide à toutes les données.

B Le Software

Le **système d'exploitation** (*OS : Operating System*) est un ensemble de programmes qui fait le lien entre les logiciels (ou les programmes que vous rédigerez) et les éléments matériels de votre ordinateur. Il permet l'échange entre le processeur, la mémoire et les systèmes applicatifs (voir la figure 5).

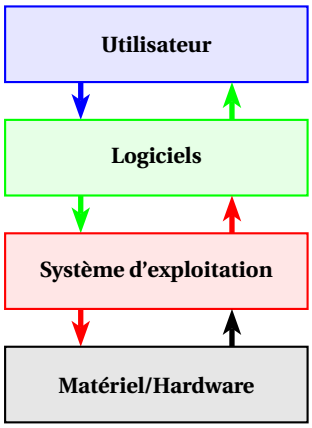


Figure 5: Rôle du système d'exploitation

Il est démarré directement à partir d'un secteur d'amorçage après le BIOS.

Son rôle est de :

- gérer le processeur, en allouant ses ressources aux différentes applications (il peut gérer les priorités, faire fonctionner des tâches en parallèle...),
- gérer les accès mémoire (RAM et ROM) et le système de fichiers,
- gérer les droits d'accès, lecture, écriture, exécution,
- servir d'interface à la manipulation des périphériques (évite au programmeur de donner des instructions qui seraient propres à chaque périphérique). Pour cela il fait appel aux *pilotes* (pour traduire les instructions en fonction du matériel),
- Détecte et récupère les erreurs et si besoin les signale à l'utilisateur,
- ...

Il est composé

- du **noyau** (kernel) qui contient les fonctions fondamentales nécessaires au bon fonctionnement de l'ordinateur.
- le **shell** (coquille) pour communiquer directement avec le noyau en lignes de commande.
- le **système de fichiers** (*FS : File system*) qui organise les fichiers en arborescence.

Remarque : Un ordinateur peut fonctionner avec différents systèmes d'exploitation. Il faut simplement que l'OS soit programmé pour savoir contrôler l'architecture matérielle et ses composants. Il est par exemple très simple d'installer un système d'exploitation Linux sur une machine Windows.

L'**arborescence fichier** est gérée par le système d'exploitation. Chaque système d'exploitation gère ses arborescences de manière spécifique.

Le système de fichiers permet de situer physiquement sur la mémoire le contenu du fichier. Dans la mesure du possible, l'OS veillera à placer un même fichier sur des zones mémoire contigües, cela évitera d'avoir à aller le chercher en plusieurs endroits du disque et accélèrera les processus. Cette ambition reste néanmoins utopique en raison des modifications de fichiers, suppressions... qui ont vite fait de mettre le bazar dans le disque. Un disque où les fichiers sont en "petits morceaux" éparpillés est dit *fragmenté*. De plus en plus, les systèmes d'exploitation intègrent des processus pour défragmenter le disque au fur et à mesure de son utilisation[5].
Comme chaque système gère les fichiers à sa façon, un disque n'est pas structuré de la même manière selon l'OS (et son utilisation), c'est la raison pour laquelle il existe différents formats[6].

- NTFS pour Windows, et ReFS qui se développe,
- HFS+ pour Mac,
- ext4 et bientôt Btrfs pour Linux,
- FAT32 développé par Microsopft et souvent utilisé pour les échanges (clés USB). C'était le format utilisé de Windows 95 jusqu'à Windows Millénium.
- ...

Permissions d'accès aux fichiers et dossiers
Chaque fichier (ou dossier) est affecté de certaines permissions pour définir qui a le droit de le lire, de le modifier ou de l'exécuter. En général, vous n'avez pas besoin de vous préoccuper de la gestion des droits car le système d'exploitation les gère pour vous. En revanche, si vous êtes amené à mettre des ressources en ligne, la gestion des droits d'accès deviendra une préoccupation essentielle pour la sécurité de votre site et de ses données.

Pour un fichier (ou un dossier), les différents droits sont :

- la lecture (**r** *read*) pour lire le contenu du fichier,
- l'écriture (**w** *write*) pour écrire ou modifier le fichier,
- l'exécution (**x**) pour exécuter le fichier (par exemple un script ou un logiciel).

Attribuer des droits à un fichier, c'est associer à chaque catégorie d'utilisateur certains droits (lecture, écriture, exécution).
Les catégories d'utilisateur sont :

- le propriétaire du fichier désigné par la lettre **u** (*user*),
- le groupe du propriétaire désigné par la lettre **g** (*group*).
- les autres désignés par la lettre **o** (*other*),

On peut indiquer les permissions d'un fichier en mettant à la suite les droits de chaque catégorie. Par exemple si le fichier `file` est affecté des permissions `rwxr-x--x`, cela veut dire que le propriétaire à tous les droits (lecture, écriture et exécution), que le groupe peut lire et exécuter le fichier et que les autres ne peuvent que l'exécuter.
Les permissions peuvent aussi être traduites numériquement. À chaque droit correspond un nombre :

- 4 pour la lecture
- 2 pour l'écriture
- 1 pour exécution

Les différents droits sont additionnés pour donner le code. Avec l'exemple précédent, le propriétaire a tous les droits : $4+2+1=7$, le groupe a la lecture et l'exécution : $4+1=5$, et les autres n'ont que l'exécution : 1.
Les droits du fichier s'écrivent alors 451[7].
Remarque : En général, il existe aussi un super utilisateur, ou administrateur qui a tous les droits. Il est désigné par `sudo` ou `root`. Les systèmes d'exploitation récents évitent de donner les droits du super utilisateur sur une session normale pour limiter les risques d'erreur pouvant endommager le système.

[5] En raison de ces évolutions, la défragmentation manuelle n'est par exemple plus accessible directement depuis Windows 8, bien qu'elle soit encore possible par le shell.
[6] Ces formats ne sont pas liés au système physique qui est commun à tous les OS. Ainsi un même disque dur peut-être formaté dans différents formats à l'aide d'un logiciel.
[7] Cela correspond à une écriture en base 8. On peut vérifier simplement que chaque type de droit donne un unique nombre.

3 PROGRAMMATION ET ALGORITHMIQUE

A Qu'est-ce qu'un langage de programmation

Les langages de programmation sont au fondement de toute l'informatique. Ils servent à communiquer avec la machine pour lui donner une suite d'instructions à réaliser (faire un calcul, afficher quelque chose...)

À la base, un ordinateur ne code que des 0 et des 1 sur son disque dur, sur un CD... C'est un codage qui est facile à réaliser mécaniquement. Par exemple sur une plaque, un zéro est représenté par un trou, et un 1 par une bosse.[8]

Chaque information que nous allons donner à un ordinateur devra être transformée en 0 et en 1 pour pouvoir être comprise par l'ordinateur. Vous imaginez bien que si on devait tout écrire nous-même en 0 et en 1, ce ne serait pas très pratique !

Un langage de programmation sera donc simplement une langue compréhensible par l'humain et qui sera ensuite traduite en langage machine.

> Il existe des langages de plus ou moins haut niveau. Un langage de haut niveau permet de faire abstraction de beaucoup de problématiques purement informatiques (gestion de la mémoire...) pour se concentrer sur les instructions.
> Python fait partie des langages de haut niveau.

Il existe deux[9] types de langages, les langages compilés et les langages interprétés (voir figure 6).

> En réalité, de plus en plus de langages dits interprétés sont des **langages compilés en langage intermédiaire bytecode**. Cela signifie qu'au moment de l'interprétation, une compilation a lieu en bytecode. Le bytecode est une forme de langage intermédiaire entre le code source et le langage machine.
> C'est le cas pour Python, et le bytcode nécessite Python pour être interprété mais est plus rapide à exécuter que le code source. L'étape de compilation n'est pas visible pour l'utilisateur, elle produit le fichier d'extension `.pyc`.

Lorsque vous avez un logiciel comme Word, Firefox,..., il s'agit d'un programme compilé : si vous ouvrez le fichier du programme (fichier.exe sous Windows), il n'est pas compréhensible : vous pouvez simplement l'exécuter. Ceux qui programment le logiciel possèdent le texte du programme, mais il ne vous vendent que le programme déjà compilé.
Pour vous donner des idées, un système d'exploitation comme Windows, est un programme en plusieurs morceaux qui utilise plusieurs langages de programmation, mais majoritairement du C/C++. Windows XP contenait 40 millions de

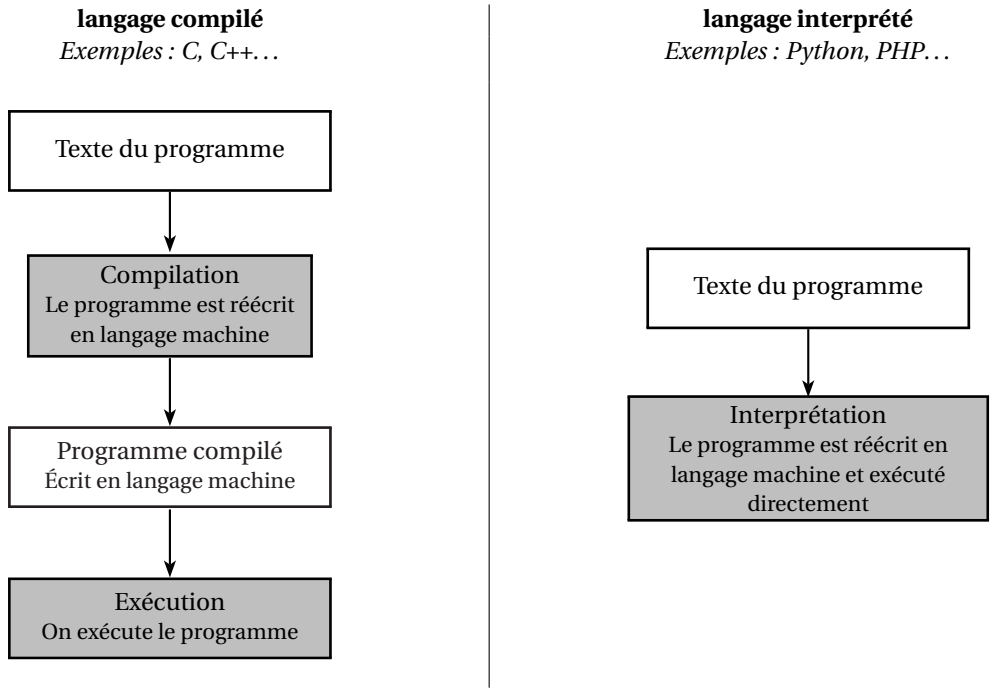


Figure 6: Langage compilé - langage interprété

[8] Le principe de creux et de bosses est utilisé pour les CD. Pour les disques durs, on utilise plutôt le magnétisme comme expliqué plus haut : de façon simplifié, l'ordinateur regarde si l'aimant est orienté dans un sens ou dans l'autre et en déduit la valeur du *bit* : 0 ou 1.

[9] En fait, cela dépend de l'implémentation du langage plus que du langage lui-même et un langage de type interprété pourrait aussi être implémenté pour fonctionner avec un compilateur. Néanmoins, il existe un usage majoritaire pour chaque langage ce qui explique la dénomination.

lignes de code. Pas étonnant qu'il y ait des erreurs et des failles !

Il existe plus de 7000 langages de programmation différents, mais peu sont effectivement utilisés. Le fonctionnement de chacun est spécifique et répond à des objectifs particuliers. Un programmeur connaît toujours plusieurs langages et utilise l'un ou l'autre en fonction de ce qu'il veut faire.

> HTML qui "code" les pages web n'est pas un langage de programmation. Il ne permet de donner aucune instruction à l'ordinateur. C'est juste une convention pour dire que tel élément est un titre, tel autre est un paragraphe...
> Chaque navigateur lira ce "code" et choisira comment l'afficher. Mais il n'est pas possible de faire faire des choses intelligentes à HTML. C'est simplement du texte mis en forme suivant certaines conventions.
> Par contre, il existe PHP qui est un langage de programmation pour le web. Lorsque vous allez sur une page web, vous ne voyez pas de PHP parce qu'il a déjà été exécuté par le serveur et vous ne recevez qu'une page HTML toute simple (ou presque).

Nous allons apprendre les langages de programmation un peu comme on apprend des langues étrangères. Nous apprendrons leur syntaxe et leur grammaire pour pouvoir communiquer avec la machine.

Pour écrire un programme, il faut être précis et rigoureux. En effet, à la moindre faute de grammaire, l'interpréteur ne comprend plus rien et renvoie un résultat faux ou une erreur. Il est encore moins tolérant que votre prof.

B Algorithmique

Maintenant que vous savez ce qu'est un langage de programmation, il faut apprendre à l'utiliser de façon intelligente. En effet, il ne suffit pas d'apprendre un langage, il faut aussi savoir ce que l'on veut dire avec ce langage. Que veut-on que fasse l'ordinateur ? C'est l'objet de l'**algorithmique**.

Le langage concerne la forme[10]
L'algorithmique concerne le contenu

Lorsque vous avez un article de journal, vous pouvez avoir le même article en plusieurs langues différentes, mais le contenu sera le même.

L'algorithmique que vous apprendrez, c'est "le contenu du journal". Vous pourrez donc l'utiliser ensuite sur n'importe quel langage de programmation, et pas seulement avec Python.

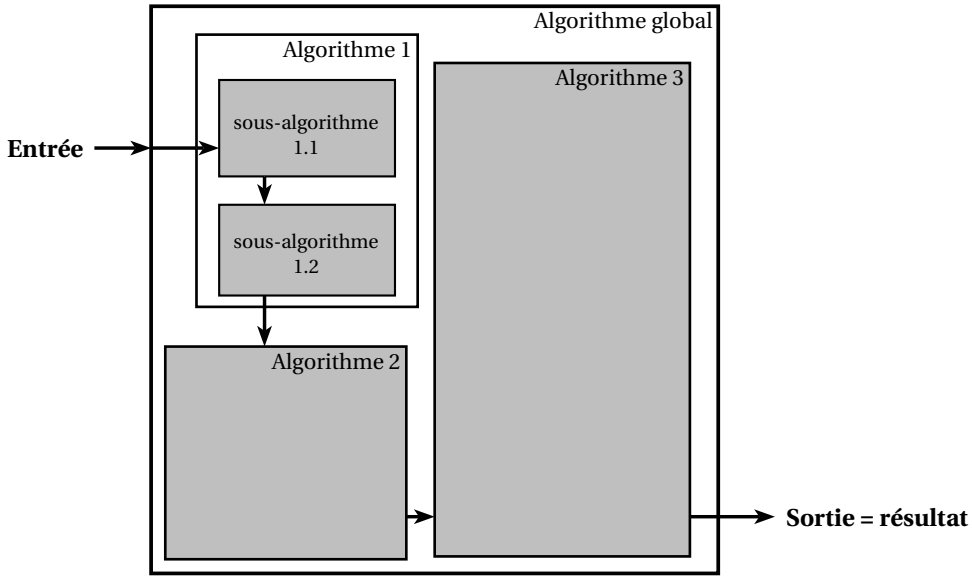
Vous imaginez bien que lorsque vous réalisez de gros programmes, de plusieurs milliers, voire millions de lignes, il n'est pas possible de mettre un programmeur qui commence à la première ligne et qui avance jusqu'à la dernière ligne comme s'il écrivait un livre. C'est la raison pour laquelle les programmes sont toujours écrits par "**blocs**" que l'on imbrique les uns dans les autres, ou que l'on accole... comme dans un jeu de construction. Chaque programmeur est alors responsable du codage de certains blocs.

Les algorithmes sont donc des structures naturellement **récursives**. Cela veut dire que chaque algorithme[11] peut être décomposé en plus petits algorithmes (qui peuvent eux-mêmes être décomposés à nouveau, et ainsi de suite). Réciproquement, chaque algorithme que j'ai écrit peut ensuite servir de bloc pour un algorithme "plus gros" qui l'englobe.

> Dans la réalité, ce schéma (figure 7) peut être implémenté de deux façons selon le type de langage.
>
> - le langage **fonctionnel** crée les sous-algorithmes qu'il réutilise un maximum de fois dans des "méta-algorithmes"... C'est ce que nous avons proposé ici (et sans doute le plus simple) en partant des briques élémentaires pour construire l'édifice.
> - le langage orienté **objet** possède une logique plus algébrique. On crée des *objets* à l'intérieur de *classes*. Chaque objet possède des *attributs* et des *méthodes*. On part de l'objet le plus général possible que l'on spécifie ensuite par un système d'*héritage* en fonction de l'utilisation voulue. La logique est donc un peu inversée, puisque l'on part du plus général pour aller vers le particulier. Ce type de programmation est privilégié aujourd'hui, même si nous l'utiliserons peu cette année.
>
> Python permet de programmer suivant les deux méthodes.

Par la suite, chacun de ces blocs fonctionnera un peu comme une boîte qui "transforme" des données d'entrée en données de sortie.
On définira alors un algorithme de cette façon :

[10] En réalité, comme pour une langue, la forme va influencer le contenu. C'est la raison pour laquelle il existe une multitude de langages de programmation différents.
[11] sauf un algorithme *élémentaire*



Les zones grisées peuvent être décomposées à nouveau

Figure 7: Schéma "digestif" d'un algorithme

Définition 3.1 :
Un algorithme est un processus logique qui comporte une entrée, un traitement, et une sortie.
Un algorithme comporte un nombre fini d'instructions et **se termine en un temps fini**.

Remarque : On pourrait remplacer le terme "logique" par "déterministe" dans la définition. Cela signifie d'une certaine manière que l'algorithme n'a pas de libre arbitre, qu'il n'improvise pas. Quand je lui donne les mêmes données à l'entrée, il me donnera toujours les mêmes données à la sortie : il n'y a pas d'aléatoire[12].

Comme spécifié plus haut, on pourrait parler d'état initial et d'état final au lieu d'entrée et de sortie.

Lorsque l'on veut écrire un programme, le plus important est de définir proprement quelles seront nos "briques", ce qu'elles recevront en entrée et le résultat qu'elles doivent donner en sortie. Lorsque ce travail est fait, il ne suffit plus que d'écrire les algorithmes pour chaque brique.

Avant d'écrire un algorithme, posez vous toujours ces questions :
Quelle entrée vais-je donner à mon algorithme ?
Quel résultat doit-il me fournir ?

C Bonne exécution d'un programme

Pour définir un algorithme, nous avons parlé de processus logique. Écrire un algorithme relève donc essentiellement du domaine mathématique dont la logique est partie intégrante. La partie informatique de la programmation, n'est plus tellement l'algorithme lui-même, mais plutôt son écriture avec le bon langage de programmation.

Mathématiques *"Contenu"*	**Informatique** *"Forme"*	
Création de l'algorithme →	Écriture de l'algorithme avec le langage de programmation choisi →	Programme complet
On s'assure qu'il renvoie bien le résultat demandé pour toutes les entrées possibles	On choisit un langage de programmation, et on écrit l'algorithme avec ce langage	

Cela est le signe d'une exigence : s'assurer qu'un algorithme se termine forcément et qu'il donne le résultat voulu dans tous les cas.

[12] Les ordinateurs ne savent pas générer d'aléatoire. Même lorsque vous demandez un nombre aléatoire à un ordinateur ou à une calculatrice, ce nombre n'est pas vraiment aléatoire, mais est calculé de façon suffisamment complexe pour qu'il en donne l'impression. Cela peut parfois poser des problèmes lors de certaines études pointues qui demandent des nombres *vraiment* aléatoires.

Par exemple, si un algorithme prend en entrée un nombre quelconque, il faut bien vérifier que cela marche pour tous les nombres (positifs, négatifs, nuls, entiers, décimaux...). Ainsi, avant de diviser par ce nombre, je m'assurerai qu'il n'est pas nul.

Normalement, tout algorithme doit pouvoir être prouvé (comme un théorème mathématique). C'est un exercice formel long et un peu compliqué. Dans la réalité, les algorithmes sont rarement prouvés, car cela serait trop long à mettre réaliser. Nous ne le ferons pas non plus.

En pratique, on remplace donc les preuves par des **tests**. Ces tests ne pourront jamais nous prouver que l'algorithme est *juste*, mais ils nous aideront à éviter de nombreuses erreurs. Cela suppose simplement de penser à faire des tests qui recouvrent le maximum de cas possibles (nombres positifs et négatifs, nuls...) et de vérifier à chaque fois que le résultat rendu est le bon.
Il ne suffit pas d'obtenir un résultat, il faut s'assurer que ce soit celui voulu. On commencera donc par des tests dont on peut prévoir l'issue sans utiliser l'ordinateur.

De même, dès que vous programmerez une boucle, il faudra toujours vous demander si elle se termine dans tous les cas et ne risque pas de tourner indéfiniment.

D Algobox

Algobox est un langage de programmation à usage pédagogique uniquement. La syntaxe (la "grammaire") de ce langage est beaucoup trop lourde pour que l'on puisse rédiger de gros programmes avec.

L'avantage de ce langage est qu'il met bien en exergue l'idée d'un algorithme construit à partir de boîtes (box en anglais).

Dès que vous avez bien compris cette structuration des algorithmes comme des boîtes ou des briques que l'on assemble, il est temps de passer à un langage un peu plus sérieux. C'est ce que nous faisons avec Python.

E Python

Python est un langage qui a été conçu vers 1990 par Guido van Rossum. Son nom vient de la série télévisée "Monty Python" dont le créateur était un passionné.

C'est un langage assez simple, bien documenté et donc adapté pour débuter. Mais il est également très utilisé par des programmeurs expérimentés. Si vous êtes amenés à travailler avec un langage de programmation en entreprise (dans un avenir pas trop lointain...), il y a de fortes chances que ce soit celui-ci.

Son succès, fait qu'il est enrichi de nombreuses bibliothèques (des collections d'algorithmes pour réaliser des opérations qui ne sont pas *natives* à Python : travail sur les tableaux, tracer des graphes, réaliser des dessins dynamiques...).