Automatisation du traitement de l'information : une évolution des capacités humaines

Introduction :

Les systèmes de traitement automatisés sont omniprésents dans notre vie de tous les jours, sous des formes très variées : ascenseurs, distributeurs de boissons, barrières de péage, machines à laver, ordinateurs, smartphones... Ils nous rendent de nombreux services, sans que l'on n'y prête toujours attention. C'est surtout quand ils se mettent à dysfonctionner qu'on réalise leur utilité et leur fiabilité au quotidien.

Tout d'abord, nous présenterons le développement historique des automatismes et l'émergence du traitement automatisé de l'information. Nous étudierons ensuite le fonctionnement de ces systèmes, et enfin leurs possibles dysfonctionnements.

1 Développement de l'automatisation

Si les automatismes et les systèmes de traitement automatisés font partie de notre quotidien moderne, leurs racines sont assez anciennes.

a. Contexte historique

La notion de **procédure** visant à traiter un problème remonte à l'Antiquité. Les mathématiciens grecs proposent des ensembles de règles formelles pour résoudre différents problèmes.

Le terme **algorithme** qui les décrit est inventé au IX^{e} siècle par le mathématicien perse Al-Khwârizmî.

Définition

Algorithme :

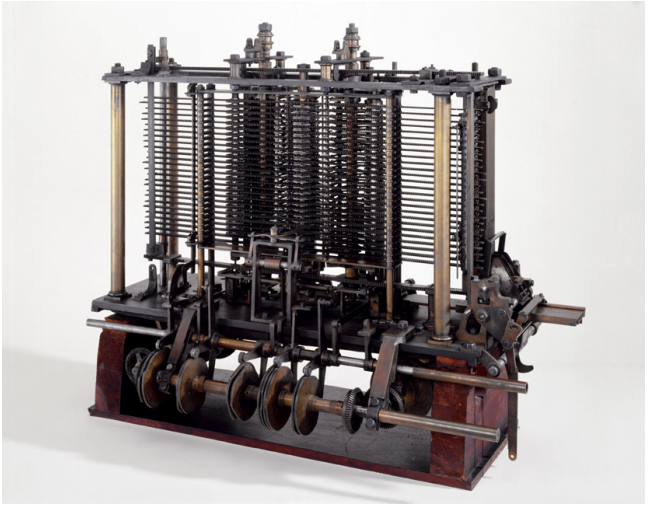
> Un algorithme désigne une suite finie et ordonnée de règles opératoires à suivre en vue de résoudre un problème ou d'obtenir un résultat.

Astuce

> C'est le nom latinisé de Al-Khuwârizmî, à savoir Algorismi, qui donne par la suite le mot « algorithme ».

Au cours de l'histoire, l'être humain construit toutes sortes d'outils et de machines, qu'il améliore et perfectionne au fil du temps.
Les premières machines à calculer, ancêtres des calculatrices modernes, voient le jour dès le XVII^{e} siècle.

La notion de **programme informatique** apparaît au XIX^{e} siècle, proposée par **Ada Lovelace** pour la **machine analytique de Charles Babbage**.



Ada Lovelace est une pionnière : étant l'auteur du tout premier programme informatique ainsi formalisé, elle est considérée comme la toute première informaticienne au monde, ouvrant la voie à ce qui deviendra plus tard la **science informatique**.

→ Ces bases historiques seront le socle sur lequel les machines mécaniques deviendront progressivement au XX^{e} siècle des ordinateurs et des systèmes automatisés de plus en plus puissants et sophistiqués.

À retenir

> Le mathématicien britannique **Alan Turing** conceptualise en 1936 un outil abstrait universel, appelé **« machine de Turing »**, fonctionnant selon une logique générale qui serait applicable à toutes sortes de calculs et de traitements.
> Cette machine universelle théorique est capable de recevoir des données et d'y appliquer une séquence de traitements préalablement définis sous forme d'instructions élémentaires.

Le caractère universel de la logique de traitement imaginée par Turing sert de base au mathématicien et physicien américano-hongrois **John von Neumann** qui conceptualise en 1945 un ordinateur stockant un programme en **mémoire**.

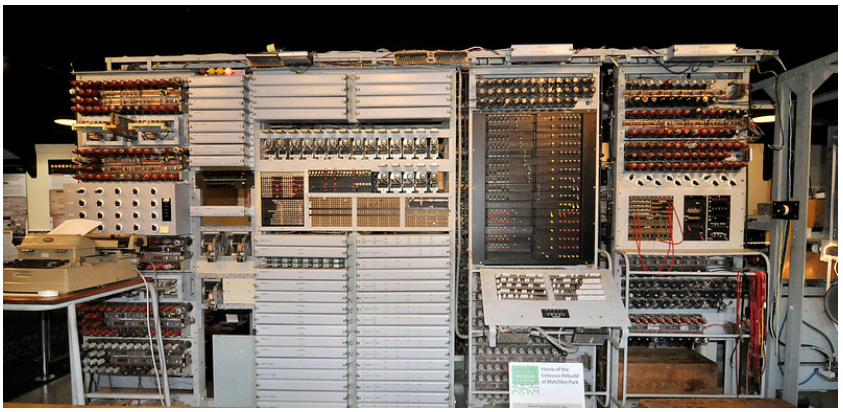
L'architecture de l'ordinateur de John von Neumann s'appuie sur six éléments :

- une **unité de traitement** composée d'une unité arithmétique et logique et de registres permettant d'effectuer des calculs ;
- une **unité de contrôle**, partie centrale qui pilote les séquences d'instructions ;
- une **mémoire de travail ou mémoire vive** qui stocke les programmes et les données de travail ;
- une **mémoire de stockage externe**, moins rapide en lecture et écriture que la mémoire de travail, et qui sert à conserver durablement les programmes et données de la machine ;
- des **mécanismes d'entrées-sorties** qui constituent les interfaces de communication entre l'ordinateur et son environnement extérieur.

Le système de calcul numérique automatique qu'il imagine est ainsi capable d'effectuer des traitements, de stocker des données, et d'interagir avec son environnement par des mécanismes d'entrées-sorties.

→ L'architecture de John von Neumann aboutit à la fabrication d'ordinateurs, construits d'abord avec des interrupteurs, des relais et des tubes à vide. Le **stockage des données** est effectué sur des cartes ou des rubans perforés.

À cette époque les ordinateurs occupent des pièces entières, consomment beaucoup d'électricité et disposent d'une capacité de traitement très limitée.



L'apparition des **transistors** dans les années 1950 permet de fabriquer des ordinateurs moins volumineux.
Les premiers circuits intégrés ou **puces électroniques** apparaissent dans les années 1960, ouvrant la voie à la **miniaturisation des composants informatiques**.
Les ordinateurs commencent à tenir dans des armoires, puis dans des coffrets de taille réduite.
Dans les années 1980, la miniaturisation a grandement progressé et les micro-ordinateurs de bureau équipés de **micro-processeurs** font leur apparition, bientôt suivis par les premiers modèles transportables, avec une capacité mémoire toujours croissante, aussi bien pour la mémoire vive (mémoire de travail) que l'espace mémoire de stockage (disquettes puis disques durs).
Les ordinateurs modernes traitent indifféremment toutes sortes de données (textes, images, sons, etc.).

À retenir

> La miniaturisation et l'augmentation de puissance des processeurs se poursuivent au cours des décennies suivantes, permettant aujourd'hui d'intégrer de puissants systèmes informatiques dans des objets aussi compacts qu'une tablette, un smartphone ou une montre connectée.

b. Importance des systèmes automatisés dans la vie courante

À retenir

> Les systèmes automatisés sont très présents dans notre cadre de vie moderne, aussi bien dans les domiciles que dans les entreprises ou dans l'espace public.

Certains sont relativement rudimentaires, d'autres très sophistiqués. Ils contribuent à l'amélioration de notre vie quotidienne et nous comptons beaucoup sur eux pour nous rendre toutes sortes de services.

Quelques énumérations non exhaustives nous permettent de réaliser à quel point les systèmes automatisés sont présents dans notre environnement.

- Nos domiciles peuvent être équipés d'ordinateurs, de tablettes, de box Internet, de téléviseurs, de thermostats connectés, de lave-linges, de portes de garage télécommandées…

- Les entreprises sont équipées de nombreux systèmes de traitement automatisés : outre les outils informatiques aujourd'hui généralisés, les lignes de production comportent de nombreux automates, comme dans la production automobile où différents robots contribuent à l'assemblage et au contrôle de qualité des véhicules.

- Dans l'espace public, on trouve des bornes d'accueil multimédia, des ascenseurs, des escalators, des panneaux d'affichage dynamiques, des feux de circulation, des systèmes d'éclairage public pilotés, des distributeurs de billets de banque, d'avion ou de train…
  Les circulations maritimes, ferroviaires et aériennes s'appuient sur différents systèmes automatisés d'information et de régulation du trafic. Ces systèmes pourraient être amenés à fortement se développer à l'avenir pour accompagner l'essor de véhicules autonomes nécessitant de nombreux échanges entre systèmes automatisés.

→ Malgré la grande diversité de leurs usages, les systèmes de traitements automatisés reposent sur des bases identiques.

2 | Systèmes automatisés de traitement d'information

Un ascenseur, un smartphone et une machine à laver sont *a priori* très différents. Pourtant, au-delà des différences d'aspect et d'usage, leurs architectures et leurs modes de fonctionnement sont très proches.

a. Architecture d'un système automatisé

À retenir

Un système automatisé est composé de deux parties distinctes :

- la **partie commande**, qui traite des informations ;
- la **partie opérative**, composée d'éléments d'interface.

→ La partie commande est comparable au cerveau humain : c'est elle qui contient les informations et fournit les instructions au reste du système.

→ La partie opérative permet l'interface avec le monde réel. Dans le cas d'automatismes d'industriels, cette partie peut se composer de **capteurs** et d'**actionneurs**.

Définition

Capteur :

Un capteur est un dispositif qui transforme une grandeur physique (température, pression, longueur, etc.) en signal, le plus souvent électrique ou électronique.

Exemple

Un ascenseur est doté de cellules photo-électriques pour détecter par l'interruption d'un faisceau lumineux le fait qu'une personne ou quelque chose se trouve au niveau de la porte.

La partie commande ne déclenchera la fermeture de la porte de l'ascenseur que si le faisceau lumineux émis est bien reçu par le capteur.

Définition

Actionneur :

Un actionneur est un dispositif capable de produire un phénomène physique (déplacement, émission d'un son, etc.) en fonction d'un signal reçu.

Exemple

Dans une machine à laver, une électrovanne régule l'arrivée de l'eau pour en fournir uniquement aux moments prévus par le cycle de lavage défini dans la partie commande.

Il existe de nombreux types de capteurs et d'actionneurs afin de permettre les différentes manipulations physiques dont doivent être capables les systèmes automatisés.

Quand le système de traitement automatisé est un ordinateur ou un smartphone, les **éléments d'interface homme-machine** sont communément désignés **périphériques d'entrées-sorties**.

Exemple

- Le clavier ou la souris sont des périphériques d'entrée : ils fournissent des données au système informatique.
- Une imprimante ou un écran simple sont des périphériques de sortie : ils reçoivent des données du système informatique pour les fournir à l'extérieur.
- Une imprimante dotée d'une fonction de numérisation est un périphérique d'entrées-sorties, tout comme un écran tactile.

Le fonctionnement d'un système automatisé est régi par un ou plusieurs programmes.

b. Programmes et données

Les **programmes** animant les systèmes automatisés peuvent être simples ou complexes, en fonction de la sophistication de l'appareil et de la complexité du problème à résoudre.
La programmation d'une machine à café est assurément moins complexe que celle d'un avion ou d'un véhicule autonome.

Parfois, les programmes sont prédéfinis et l'utilisateur n'a aucun contrôle. Dans certains cas, l'utilisateur a la possibilité de paramétrer et, éventuellement, de programmer lui-même tout ou partie du système automatisé auquel il a accès.

 Exemple

> Un ordinateur est doté d'un système d'exploitation et d'applications logicielles prêtes à l'emploi, mais un utilisateur averti peut développer ses propres programmes, gérer les interactions avec les périphériques d'entrées-sorties de l'appareil et effectuer différents traitements de **données**.

 Définition

> **Programme :**
>
> Un programme est une suite finie d'opérations exécutables par un ordinateur.

 Définition

> **Donnée :**
>
> Une donnée est une information ou un ensemble d'informations traitées par un programme ou fournies par un programme.

Cette distinction entre programmes et données peut être nuancée par le contexte. Un programme manipule des données. Lorsqu'un programme est manipulé par un notre programme, il devient alors donnée.

Exemple

L'extrait de code ci-après en langage Python définit une fonction déterminant la parité d'un nombre.

```
def est_pair(nombre):
    """Indique si un nombre est pair.
    Prend en entrée un nombre entier.
    Retourne un booléen (True ou False).
    """
    return nombre % 2 == 0
```

Ce programme implémente un algorithme, qui traite des données :les nombres passés en arguments à l'appel de la fonction.
Le code décrivant cette fonction est exprimé sous forme de texte en langage Python. Pour l'interpréteur Python, ce code constitue des données à traiter.
De la même manière, les fichiers, y compris des programmes exécutables comme l'interpréteur Python, constituent des données pour le gestionnaire de fichiers du système d'exploitation de l'ordinateur.

Ces systèmes automatisés sont conçus pour nous rendre des services au quotidien. Leurs concepteurs doivent faire en sorte qu'ils soient robustes et fiables afin d'éviter des dysfonctionnements.

3 Dysfonctionnements logiciels

Il arrive que des logiciels dysfonctionnent. Les conséquences sont parfois bénignes, parfois beaucoup plus sérieuses.

a. Bogues

Commençons par définir ce qu'est un **bogue** (ou *bug* en anglais).

Définition

Bogue :

> Un bogue est un dysfonctionnement causé par un défaut de conception d'un programme informatique.

La gravité d'un bogue est variable : certains bogues sont bénins et causent des soucis très minimes, comme une partie d'écran non visible ou non cliquable sur un logiciel de traitement de texte, mais des bogues critiques peuvent avoir de très graves conséquences, en particulier quand ils touchent des systèmes d'importance majeure.

 Exemple

> Le premier vol de la fusée de la fusée Ariane 5 s'est soldé par sa destruction, suite à une erreur dans son système informatique, pourtant éprouvé avec succès lors des nombreux vols d'Ariane 4.
> Mais son logiciel n'avait pas été adapté à la plus grande capacité d'accélération de la version 5. Celle-ci causa un dépassement d'entier, causant la perte du contrôle de la fusée par le pilote automatique, et sa destruction quelques secondes plus tard par le système de sécurité.

L'étude d'un cas simple nous permet d'illustrer la survenue de bogues.

b. Étude de cas

Pour illustrer de manière concrète la survenue de bogues, nous utilisons le langage Python et créons en quelques lignes de code une fonction qui calcule l'IMC (indice de masse corporelle) d'un individu.

Cet indice s'obtient en divisant la masse de l'individu (improprement appelée « poids » dans le langage courant) par le carré de sa taille. La masse est exprimée en kilogrammes et la taille en mètres.

Considérons le code Python suivant :

```
def indice_masse_corporelle(masse, taille):
    imc = masse / taille ** 2
    return imc
```

Testons le code avec quelques valeurs :

```
print(indice_masse_corporelle(65, 1.75))
# affiche 21.224489795918366

print(indice_masse_corporelle(90, 1.85))
# 26.296566837107374
```

Ce code semble fonctionner correctement. Mais il n'est pas exempt de bogues. Nous pouvons facilement le mettre en évidence.

```
print(indice_masse_corporelle(50, 0))
# affiche ZeroDivisionError: division by zero
```

Pour pallier ce bogue, on pourrait compléter le code avec un test conditionnel au début de la fonction s'assurant que la valeur de taille n'est pas nulle. De plus, on pourrait s'assurer que l'utilisateur de la fonction passe bien en argument des valeurs numériques et cohérentes avec les plages médicalement admissibles de masse et de taille.

Un bogue qui interrompt l'exécution d'un programme est fâcheux, mais cela le rend facilement repérable. Ce n'est pas toujours le cas.

Considérons le code suivant :

```
def indice_masse_corporelle(masse, taille):
    imc = masse / taille * 2
    return imc
```

Testons ce code avec quelques valeurs :

```
print(indice_masse_corporelle(65, 1.75))
# affiche 74.28571428571429

print(indice_masse_corporelle(90, 1.85))
# affiche 97.29729729729729
```

Tant que nous n'indiquons pas une taille nulle, le programme ne génère pas d'erreur. Il fournit bien un résultat pour chaque calcul, mais en y regardant de plus près, on se rend compte que les calculs sont erronés : les valeurs retournées sont très précises mais incorrectes.

Le développeur a mal implémenté la formule de calcul : par méconnaissance de la syntaxe du langage ou à cause d'une faute de frappe, la taille n'est pas élevée au carré mais seulement multipliée par deux.
En effet, la syntaxe en Python pour l'élévation à une puissance nécessite deux signes « * » consécutifs, et le code n'en comporte qu'un seul.
L'ordinateur effectue donc une multiplication par 2 au lieu de calculer le carré du nombre.

→ Notre exemple étant très simple, le bogue est assez facile à repérer. Mais dans un programme plus complexe, comportant de nombreux éléments et nécessitant de prendre en compte simultanément ou consécutivement différentes conditions, des bogues peuvent survenir.

La mise en place de tests logiciels permet de vérifier le bon fonctionnement du code.

c. Tests logiciels

À retenir

> La mise en place de tests du code constitue une bonne pratique lors du développement logiciel.
> Ces tests sont généralement appliqués à des unités individuelles de code, à l'échelle de fonctions ou de méthodes. La mise en place de tels tests unitaires est très pertinente en programmation orientée objet.

Astuce

> La programmation orientée objet s'articule autour du concept central d'objets. Ces objets définis en classes peuvent contenir à la fois des données et du code et interagir avec d'autres objets.
> Le thème de la programmation orientée objet est abordé plus en détails en spécialité « Numérique et sciences informatiques ».

Une unité peut faire l'objet de différents tests, pour vérifier que l'unité de code testée produit bien le résultat attendu en toute situation. Cela

nécessite de tester les cas d'usage courants, mais aussi d'envisager des cas plus rares, pour s'assurer que le code est apte à les gérer correctement.

Deux approches sont possibles :

- écriture du code, puis écriture des tests pour contrôler le code ;
- écriture des tests à partir desquels le code est développé.

La seconde approche est appelée **développement piloté par les tests**. Aucune ligne de code n'est écrite avant qu'au moins un test ne soit conçu. On écrit ensuite le minimum de code suffisant pour pouvoir passer le test ou les tests existants, et ainsi de suite.

Le fait de concevoir à l'avance les tests oblige à bien définir le comportement attendu du programme dans les différents cas de figure possibles. Il existe souvent plusieurs manières, parfois très différentes, d'écrire une unité de code qui réponde aux spécifications d'un ensemble de tests.

 Exemple

> On souhaite disposer d'une fonctionnalité permettant de compter les mots d'une phrase. Cela paraît relativement simple à implémenter et on pourrait être tenté de commencer immédiatement à coder, sans réaliser que cette simplicité est en réalité très relative.
>
> Sans se préoccuper de la manière dont cette fonctionnalité sera implémentée au niveau du code, on peut définir le comportement attendu pour différents cas, qui pourront être testés une fois la fonction écrite.

Chaîne entrée	Résultat attendu
`"bonjour"`	`1`
`"au revoir"`	`2`
`""`	`0`

`" "`	`0`
`" bonjour"`	`1`
`"aurevoir"`	`2`
`"bonjour."`	`1`
`"bonjour !"`	`1`
`"entendu, à bientôt !"`	`3`
`"j'ai dit bonjour"`	`4`
`"aujourd'hui"`	`1`
`"j'ai dit bonjour aujourd'hui"`	`5`

Sans avoir identifié la diversité des cas ci-dessus, on aurait pu être tenté d'implémenter la fonctionnalité en comptant simplement les espaces présents, en partant du principe qu'une phrase type comporte $n + 1$ mots par rapport au nombre d'espaces.

Mais nos cas de tests montrent que cela ne suffit pas car il peut y avoir des espaces excédentaires, du fait d'une faute de frappe ou d'une volonté de l'auteur de décaler son texte. Il faudrait remplacer tous les espaces de plus d'un caractère par un seul espace. Il faudrait aussi supprimer les éventuels espaces en début et en fin de phrase.

Cela ne suffirait toujours pas, car certains signes de ponctuation comme « : », « ! » et « ? » sont eux aussi précédés d'un espace, et ne sont pas des mots pour autant. On pourrait imaginer supprimer toutes les ponctuations, mais nos tests indiquent là encore que ce n'est pas si simple : l'apostrophe fait partie du mot « aujourd'hui » qui doit rester d'un seul tenant mais sépare « j'ai » en deux mots « j' » (je) et « ai ».

→ Cet exemple non exhaustif montre à quel point la conception des tests conduit à envisager la variété des cas de figures qui peuvent se présenter. Les tests permettent de vérifier que le code produit le résultat attendu dans

> les cas d'usage typiques mais aussi dans des cas particuliers, peut-être moins fréquents mais qui peuvent tout de même se présenter.

Certains systèmes informatiques modernes comportent des millions de lignes de code. Les protocoles de développement et de test des systèmes critiques, comme ceux utilisés en aéronautique, sont particulièrement stricts, car des dysfonctionnements peuvent avoir de très graves conséquences. Mais le recours aux tests constitue une bonne pratique pour tous les développements informatiques.

La chasse aux bogues n'est pas le seul aspect à considérer au niveau logiciel : la sécurité doit également être prise en compte pour éviter que des failles ne puissent être exploitées par des personnes malveillantes, notamment pour exploiter des donner utilisateurs, protégées maintenant par la loi.

> Conclusion :
>
> Les systèmes automatisés de traitement de l'information sont omniprésents dans notre quotidien et nous rendent de nombreux services. Après avoir présenté l'historique de leur émergence, nous en avons étudié les principales caractéristiques, au-delà des différences de formes et d'usages, pour répondre à des besoins personnels, industriels, ou d'utilité publique. Les traitements de données reposent sur des programmes, susceptibles de comporter des bogues aux conséquences plus ou moins sérieuses.
>
> Historiquement conçus de manière autonome, ces systèmes sont de plus en plus connectés à toutes sortes de réseaux, souvent de manière permanente. Cette connectivité permet d'offrir de nouveaux services. Mais le fait que cette connectivité soit indispensable au fonctionnement d'un système le aussi rend inopérant en cas de défaut de connexion. Les systèmes automatisés peuvent également être contrôlés ou interagir avec des intelligences artificielles, dont le développement fulgurant ces dernières années contribue à faire émerger toutes sortes d'usages, faisant parfois peser des risques sur nos libertés.