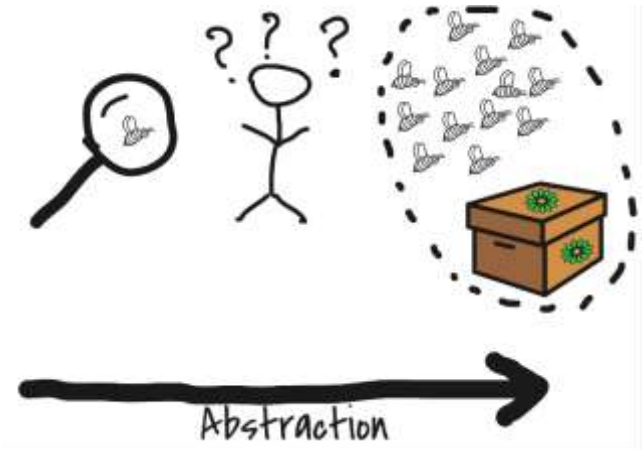
Type de document	Cours	Classe	1re	Durée	2h	Date	21/21/2024
Thème et contenu(s)	Représentation des données – Langages & programmation – Histoire – Architecture						
Capacités attendues	Comprendre comment un ordinateur représente les données simples – Savoir initialiser correctement une variable en Python – Effectuer des manipulations basiques						
Prérequis	Aucun						
Description	Cours d'introduction en classe de Première						

I) Qu'est-ce qu'un ordinateur ?

I - a) L'abstraction

Par exemple, lorsque l'apiculteur considère le comportement global de sa ruche, au lieu d'examiner précisément le comportement individuel de chaque abeille qui la compose, il fait preuve **d'abstraction**. Cela l'aide à comprendre et à anticiper l'évolution de la ruche.



L'abstraction est une notion centrale en informatique : c'est une simplification de la réalité, elle permet de mieux la comprendre

I - b) Décrire un ordinateur

Pour décrire un ordinateur, nous nous trouvons dans le même cas que l'apiculteur. Que devons-nous décrire ? On peut répondre à cette question en posant une autre question : qu'est-ce qui ne change jamais dans un ordinateur ?

- Sa construction (sa forme) ? Non. Un smartphone, une tablette, un PC portable, un PC de bureau, un PC « tout-en-1 », un Raspberry Pi, sont tous des ordinateurs très différents
- Son électronique, ses composants ? Non. Déjà nous manquerions d'abstraction, parce que chaque composant est très complexe. Ensuite, il existe une multitude de constructeurs et donc de composants, de circuits…
- Les phénomènes physiques qui se produisent à l'intérieur ? Peut-être, c'est vrai que ces phénomènes ne changent jamais. Mais les décrire reviendrait à faire une description des phénomènes électriques et électroniques, et à un niveau encore plus poussé, quantiques. Cela est très intéressant, mais pour le coup tellement abstrait que cela peut s'appliquer à tous les appareils utilisant du courant électrique !
- **Son principe de fonctionnement ? Bravo, c'est ça qui reste inchangé depuis des dizaines d'années !**

I - c) Architecture de « Von Neumann »

La paternité de cette architecture, présentée en 1945, n'est pas certaine (du moins, Von Neumann n'est pas le seul à l'avoir mise en place). L'architecture attribuée à ce scientifique Hongrois consiste à **faire abstraction du matériel** pour s'attacher aux fonctionnalités primaires d'un ordinateur.

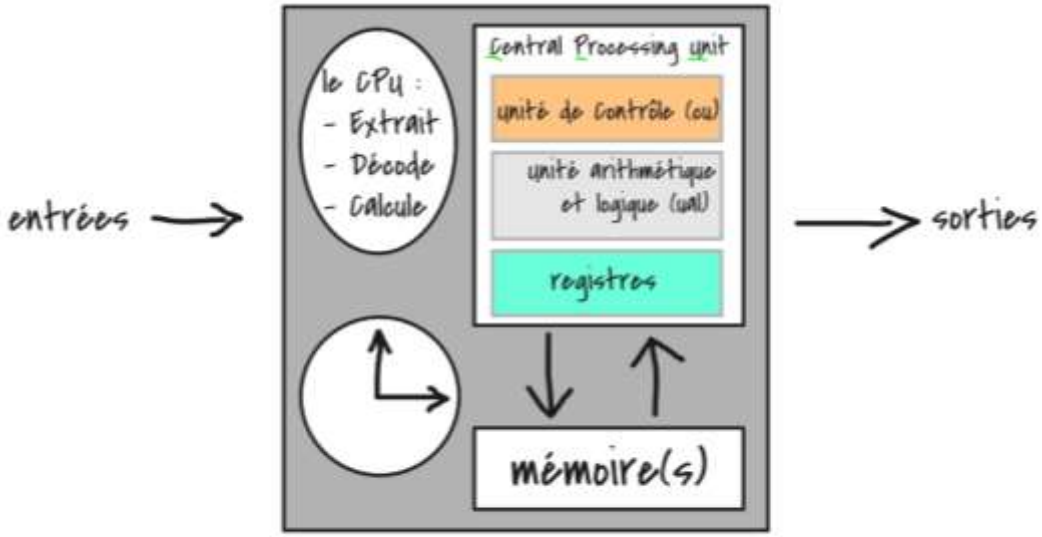
Cela peut sembler décevant la première fois qu'on l'entend, mais un ordinateur n'est rien d'autre qu'une calculatrice avec une mémoire.

La plupart des calculs qu'il fait sont simples. C'est leur enchaînement et la vitesse à laquelle ils sont effectués qui font que l'ordinateur est capable de lire de la musique, des vidéos, d'enregistrer votre voix, d'imprimer vos textes…

Le composant qui effectue les calculs dans l'ordinateur est le **CPU (Central Processing Unit)**, qu'on appelle aussi **microprocesseur** ou **processeur**. C'est le cœur de l'ordinateur, il effectue **plusieurs milliards de calculs par seconde**.

Pour faire tous ces calculs, l'ordinateur a besoin de mémoire. Elle peut être :

- **volatile,** qui s'efface lorsque l'ordinateur s'éteint, comme la **mémoire vive** ou **RAM**
- **de masse**, qui conserve les données indéfiniment, du moins tant que le matériel fonctionne. C'est le cas des **disques durs**, des **mémoires flash** (micro-mini-sd, clef USB, SSD…)



Architecture de "Von Neumann"

La mémoire et le CPU travaillent ensemble : le **programme contenu dans la mémoire vive** est récupéré, décodé puis exécuté, ligne par ligne (instruction par instruction), et ce des milliards de fois par seconde. Le CPU est composé :

- D'une **Unité Arithmétique et Logique** (UAL, ALU en anglais) qui est chargée d'effectuer les calculs
- De **registres**, qui sont des mémoires très petites et très rapides faisant partie intégrante du microprocesseur. Elles servent notamment à stocker de manière temporaires des valeurs, des adresses mémoire, des instructions…
- D'une **unité de contrôle** (UC ou CU en anglais) qui joue le rôle de régulateur. Elle contrôle, ordonne et décode les opérations entrantes, et gère les flux entre l'**UAL** et les **registres**.

Enfin, ce qui se trouve dans la mémoire n'est pas arrivé là par hasard : il y a eu à un moment donné une **entrée** dans le système (une interaction avec le monde extérieur).

De la même manière, l'ordinateur a vocation à produire les résultats de ses calculs, que ça soit à l'homme ou à d'autres machines. Il dispose donc de **sorties** (par exemple l'écran, mais aussi n'importe quoi d'autre permettant de communiquer avec l'extérieur).

Enfin, **mémoires et CPU sont munis d'horloges : l'horloge détermine la fréquence à laquelle sont effectuées les opérations**. En d'autres termes, à chaque top d'horloge une opération est effectuée, et entre deux « tops d'horloge » il ne se passe rien.

Nous venons de voir ce qu'on appelle un modèle, créé grâce à notre capacité d'abstraction.

II) Programmation : le langage machine

II - a) De la langue au langage

Nous avons rapidement parlé d'**instructions** : ce sont en d'autres termes des requêtes, le plus souvent très simples, faites au processeur et lui demandant d'effectuer des calculs. Un **programme informatique** est constitué d'instructions compréhensibles par la machine.

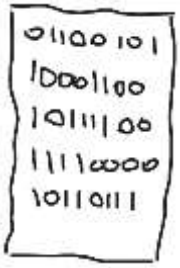
Le CPU exécute toujours les instructions une par une, dans l'ordre (séquentiellement). Un **programme** est écrit au moyen d'un **langage** : la notion de **langage** est, sous certains aspects, proche de celle de langue : tous deux possèdent une **grammaire** (syntaxe).

Un **programme** est un simple **fichier textuel** écrit dans un **langage** spécifique qui comprend plusieurs instructions. On peut lire et modifier ce fichier avec un **éditeur de texte**.

II - b) Le langage binaire

Les signaux envoyés sont des impulsions électriques (plus exactement des tensions) qui sont interprétées comme des 0 ou des 1 : il n'est donc pas faux d'affirmer qu'un ordinateur « ne sait lire » que des 0 et des 1, c'est juste un peu réducteur ☺

On appelle **langage binaire (sous-entendu en base 2)** un **langage** qui ne contient que des **suites composées de deux symboles/chiffres : 0 et 1.** Un **bit (Binary digIT)** vaut **0** ou **1 :** par commodité on regroupe les bits par **paquets de 8 : les octets** (cf. cours SNT de 2nde)



De la même manière qu'en base 10 les nombres sont composés de chiffres allant de 0 à 9, en base 2 les chiffres vont de 0…à 1 ! Les appareils électroniques évolués, sans pour autant être des ordinateurs, utilisent eux aussi le langage binaire.

II - c) Communiquer avec l'ordinateur : le langage machine

Pour communiquer avec l'ordinateur, et donc le processeur, l'être humain a abandonné depuis longtemps le langage binaire : il ne l'utilise plus que très peu, ponctuellement. Attention, car si l'humain l'a abandonné, la machine, elle, ne comprend toujours que le langage binaire !

En revanche l'homme a créé des **correspondances exactes** entre le langage binaire et un langage (ou plutôt des langages) qu'il a créé spécifiquement pour chaque type/famille de CPU : le **langage machine**, ou **assembleur** (assembly language). Ce **langage** lui sert **d'interface** pour programmer efficacement l'ordinateur, en limitant les risques d'erreurs (très élevés si on utilisait le langage binaire).

L'image suivante nous montre (à titre d'exemple !) les **instructions** (ou mnémoniques) les plus courantes en assembleur pour les CPU Intel d'une très ancienne génération (8 bits), avec leur représentation **binaire** et **hexadécimale** (nous verrons plus tard à quoi cela correspond). À chaque octet correspond une instruction (mnémonique), et vice versa.

@yatishparmar yatishparmar.com

Intel 8085 (partial) instruction set

Binary	Opcode	Mnemonic	Description
1000 0111	87	ADD A	Add the contents of register A to that of the accumulator
0011 1010	3A	LDA	Load data stored in the given memory address
0111 1001	79	MOV A C	Move data from register A to C
1100 0011	C3	JMP	Jump to instruction in specified memory address
1100 0001	C1	POP B	Pop from stack and copy to memory registers B + C

Source : Yatish Parmar, vidéo YouTube « Assembler and Instructions - A Level Computer Science »https://www.youtube.com/watch?v=x6fOb9I2NQk&ab_channel=YatishParmar

II - d) Limites de l'assembleur

Bien que constituant un progrès par rapport au langage binaire, le langage assembleur possède de nombreux inconvénients : il est complexe à apprendre et à utiliser, il est trop lié au matériel, à l'électronique et à la **logique de bas niveau**. Il demeure un **langage de bas niveau d'abstraction**. En d'autres termes :

Généralement, un langage de bas niveau d'abstraction nécessite, pour créer un programme, d'écrire plus de lignes de code que des langages de plus haut niveau.

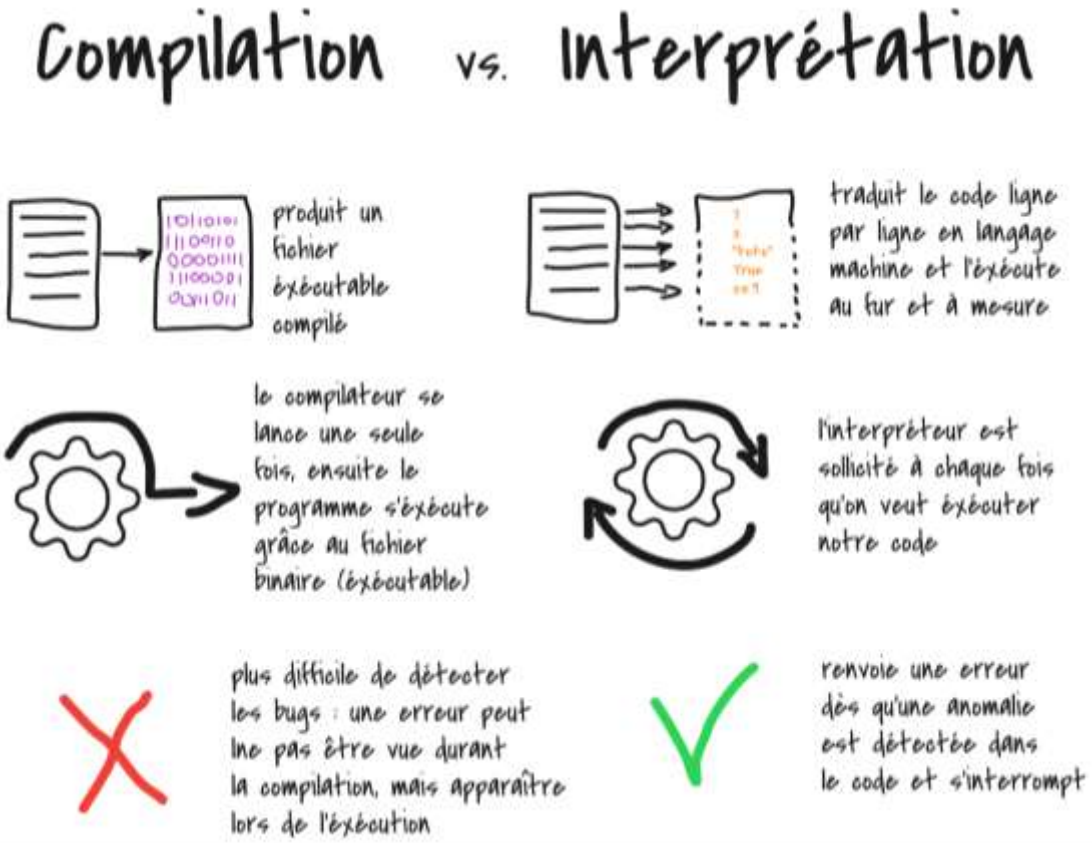
Malgré sa rapidité d'exécution, l'assembleur n'est appris et/ou utilisé que marginalement de nos jours. Mais dans certains domaines spécifiques on l'utilise encore (celui du hacking/sécurité informatique par exemple).

Pour s'extraire des contraintes matérielles du CPU, les informaticiens ont développé très tôt des langages dont l'utilisation ne requiert pas une connaissance poussée des mécanismes internes au CPU. Ces langages fournissent une **abstraction** supplémentaire par rapport au langage binaire.

III) Les autres langages de programmation

III - a) Langages compilés vs. interprétés

En plus de l'assembleur, qui détient le record du langage de programmation le plus proche de la machine, existent des milliers de langages de programmation différents.
Ils ont tous un point commun : **leur code est toujours traduit en assembleur afin de pouvoir être exécuté sur un ordinateur**. La manière dont ils sont traduits en assembleur permet de proposer une première classification de ces langages en deux familles :



- Soit l'intégralité de leur code est « lue » puis **transformée en code assembleur** placé dans un **fichier exécutable** (= « fichier binaire ») par un **compilateur** : ce sont les **langages compilés** (le plus connu est le langage C)
- Soit leur code est, ligne par ligne : lu, transformé, exécuté par un **interpréteur**. Ce sont les langages **interprétés** (par exemple PHP, Python…)

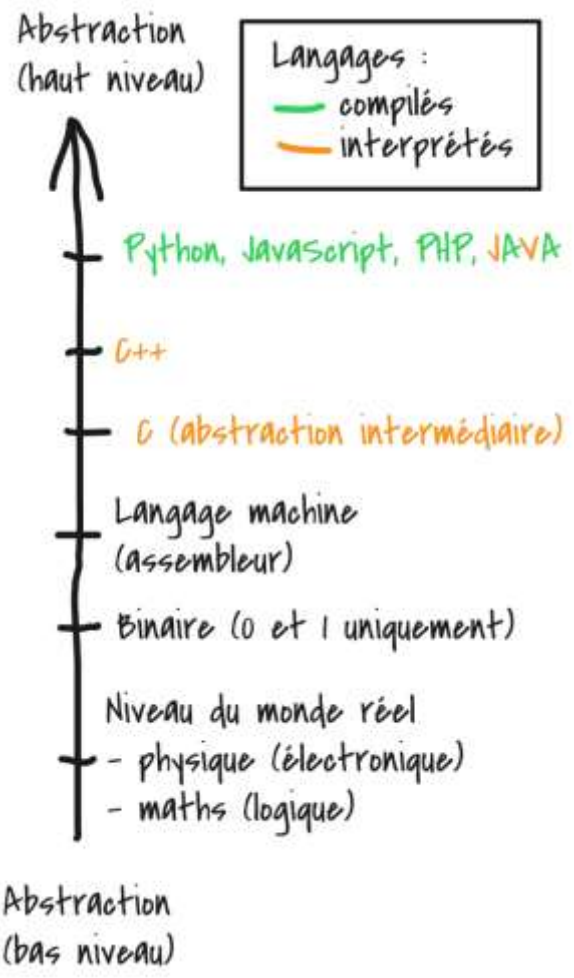
Il existe enfin des langages dont la traduction en langage machine est hybride, comme JAVA. **Les compilateurs et les interpréteurs sont eux-mêmes des programmes informatiques**

III - b) Niveau d'abstraction

Une autre manière de classer les langages est, vous l'avez peut-être deviné, par rapport à leur **niveau d'abstraction** (sous-entendu par rapport au langage binaire, qui est le moins abstrait/le plus concret du point de vue de la machine).

En général, plus un langage est de haut niveau d'abstraction, plus il est permissif, facile à apprendre, et moins l'utilisateur doit se préoccuper des aspects matériels (gestion de la mémoire…etc). Hélas, la permissivité n'est pas seulement un avantage, vous l'apprendrez bientôt à vos dépens !

Python est un langage de haut niveau d'abstraction



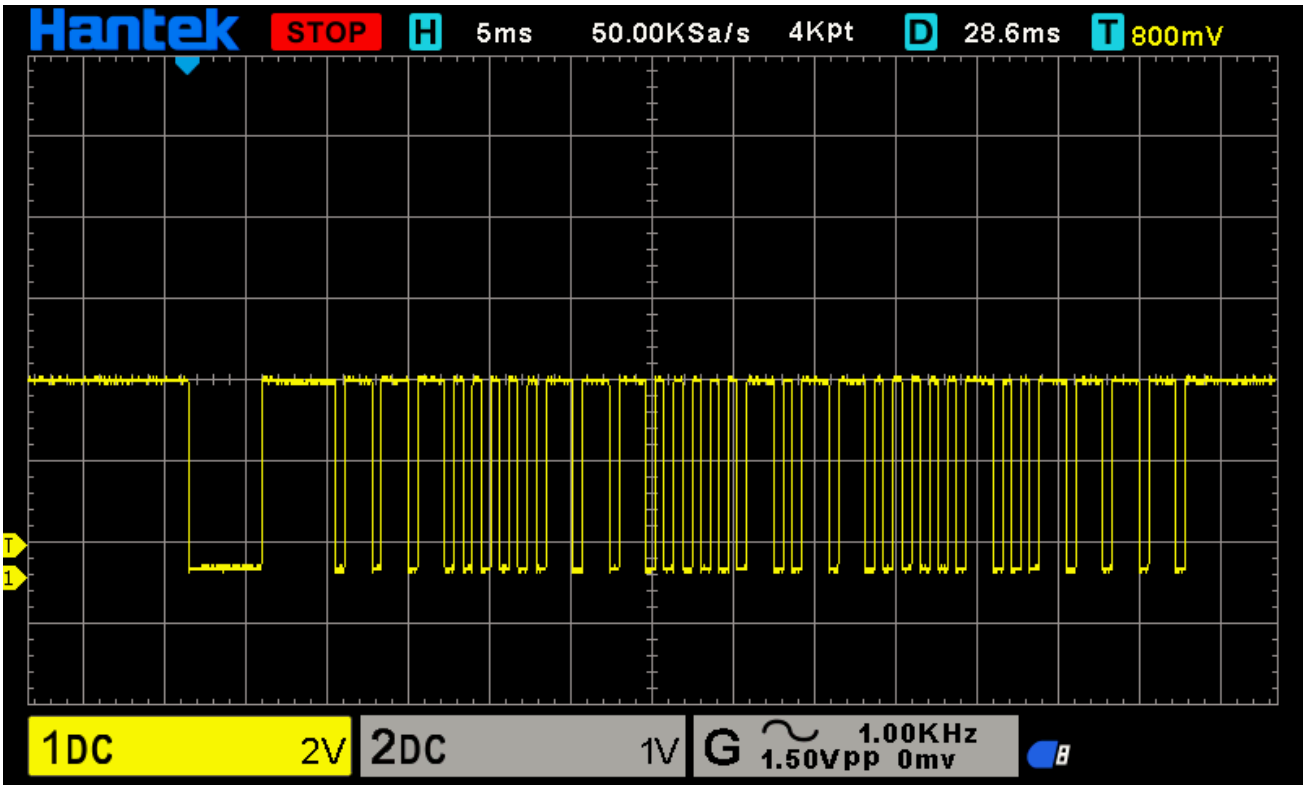
On peut comprendre ce qu'est le niveau d'abstraction d'un langage en le comparant avec n'importe quel « objet » offrant une interface pour le manipuler :

- le tableau de bord de notre voiture nous offre une interface pour actionner des circuits et des composants complexes : quand on veut mettre les warnings, on appuie une seule fois sur le bouton pour enclencher le cycle
- on n'écrit pas des lignes de code assembleur pour programmer notre machine à laver : on utilise l'interface fournie qui reste simple d'utilisation
- le clavier virtuel de notre téléphone est une interface qui nous permet d'appeler quelqu'un (on ne crée pas un programme en langage machine pour lancer un appel)
- l'interface d'une application mobile nous aide à effectuer simplement des tâches qui sont complexes pour le CPU du smartphone
- le bouton « ON » de notre ordinateur est une interface qui effectue pour nous la séquence de boot de notre ordinateur, depuis le chargement du BIOS jusqu'à celui du système d'exploitation
- À vous de trouver vos propres exemples : lancement d'une fusée, Thermomix, tout est permis !

Dans tous ces exemples, l'interface joue le rôle d'un levier qui nous permet de nous élever en termes d'abstraction

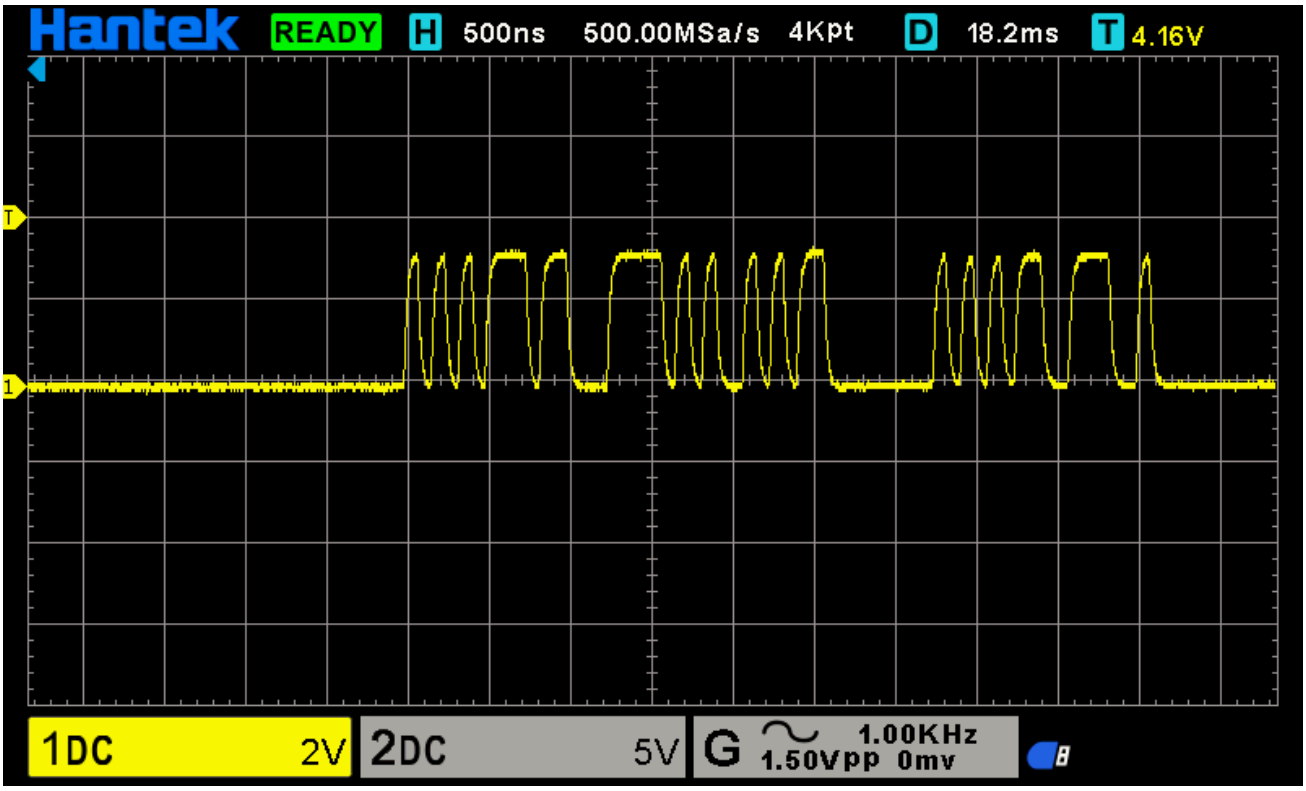
IV) 3 exemples d'utilisation du langage binaire

IV - a) Mesure du signal sortant de ma télécommande de TV



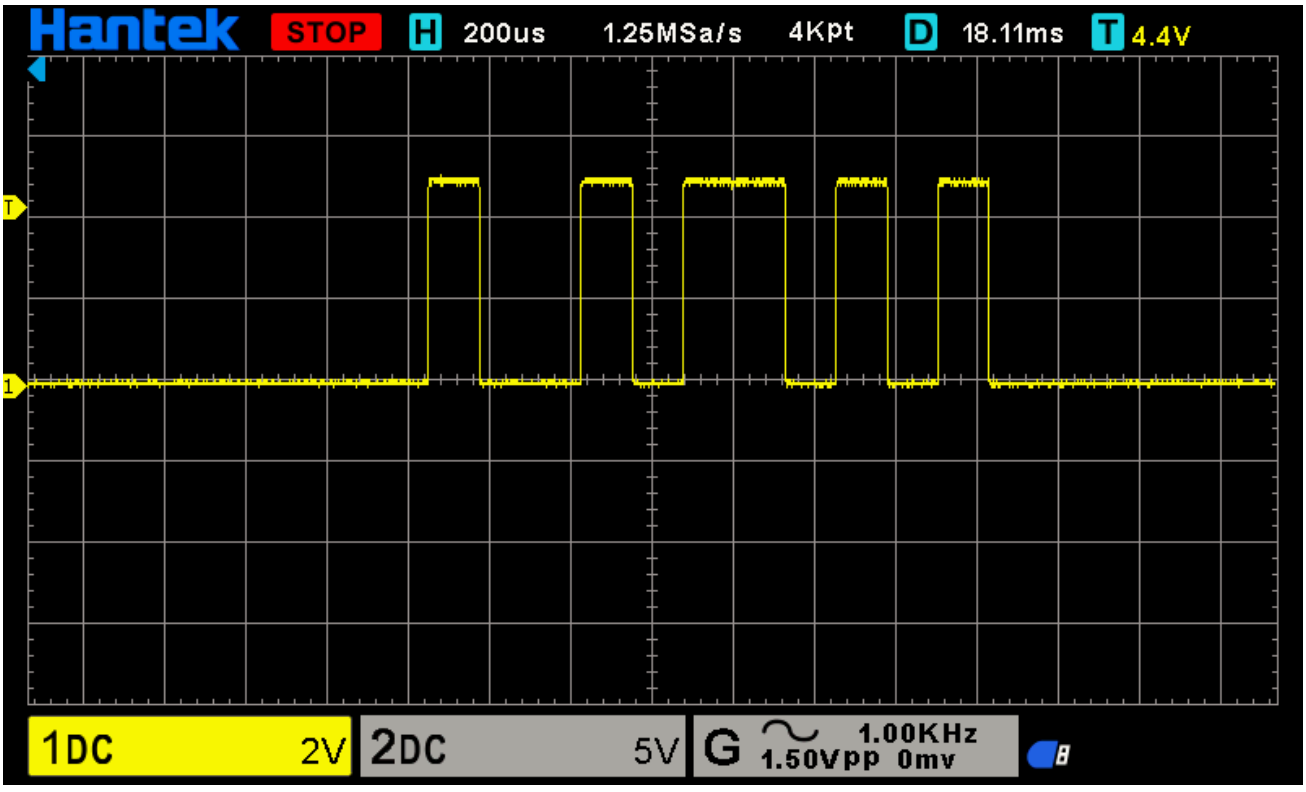
IV - b) Signal mesuré lors d'une communication USB avec un ordinateur

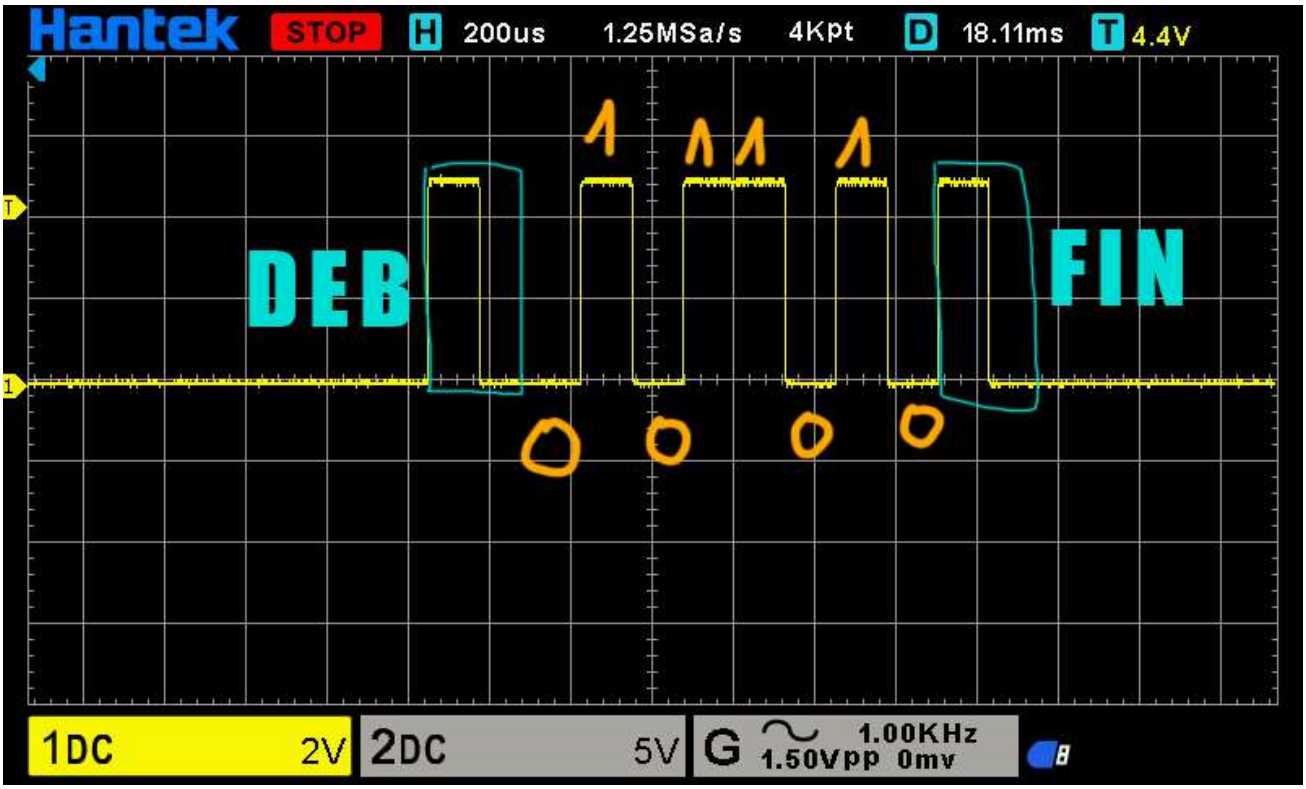
N



Notez la différence au niveau du pas de temps. C'est beaucoup plus rapide !

IV - c) Mesure d'un signal utilisant une liaison UART (émis par un Arduino)





Les octets sont transmis ici « à l'envers » (bits de poids faible en premier). L'octet transmis est donc : 01011010 ce qui correspond à l'entier 90. Sa valeur dans l'alphabet ASCII est « Z ».