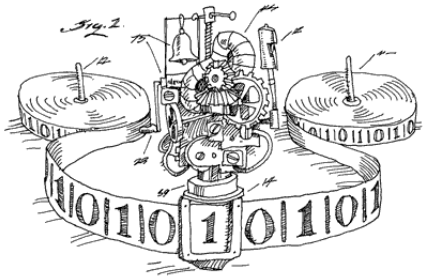
Calculabilité et décidabilité

Capacités attendues

- ✓ Comprendre que tout programme est aussi une donnée.
- ✓ Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé.
- ✓ Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.
- ✓ L'utilisation d'un interpréteur ou d'un compilateur, le téléchargement de logiciel, le fonctionnement des systèmes d'exploitation permettent de comprendre un programme comme donnée d'un autre programme.

Quelle est la limite de ce qu'on peut faire avec des programmes ?



1 Programmes et données

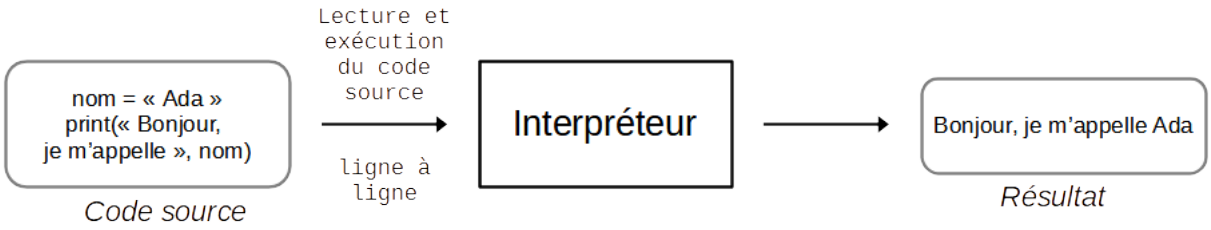
Programmes et données sont de la même nature pour un ordinateur. Les deux sont traduits en binaire et stockés de la même façon dans la mémoire. Voyons quelques exemples montrant qu'un programme peut être traité comme une donnée comme une autre, utilisée en entrée d'un autre programme.

Exemples :

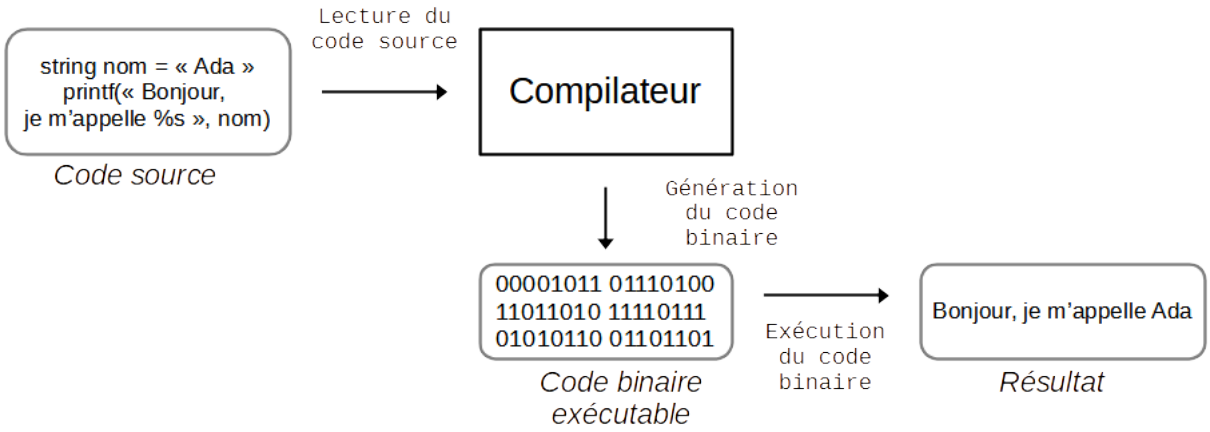
- Les **systèmes d'exploitation** sont des programmes qui gèrent les autres programmes de l'ordinateur. On peut écrire des scripts qui interagissent avec le système d'exploitation, effectuant des actions sur des fichiers, qui peuvent être exécutables (des programmes) ;
- Les **antivirus** sont des programmes qui analysent tous les fichiers de l'ordinateur, dont les exécutables.

1.1 Interpréteur et compilateur

Le code source d'un programme doit être traduit pour être compréhensible par la machine. Pour ce faire, les différents langages de programmation existant utilisent généralement un compilateur ou un interpréteur:



Interprétation d'un programme



Compilation d'un programme

> **N.B.**
> Il existe également des langages semi-interprétés qui traduisent le code source dans un langage intermédiaire.

Dans tous les cas, une analyse lexicale du programme est effectuée : le **compilateur** ou **interpréteur** identifie les éléments fondamentaux d'un programme comme les noms de variables, de fonctions, les mots-clés du langage ou bien les expressions. Il vérifie également sa structure. Le programme est donc une ***donnée d'entrée*** analysée par un compilateur ou un interpréteur.

2 Calculabilité et décidabilité

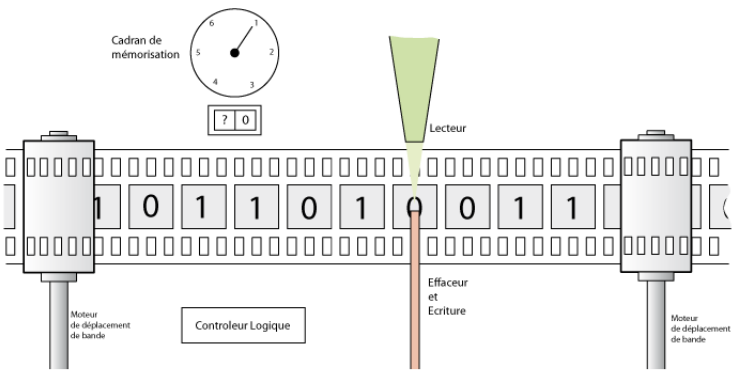
2.1 Thèse de Church

Il existe un nombre très importants de langages informatiques, dont la syntaxe diffère mais permettant d'effectuer les mêmes opérations *(hors programme : sous réserve d'être complet au sens de Turing)*. Ils ont la même puissance algorithmique, les mêmes fonctions peuvent être implémentées dans tous.

Les langages de programmation sont des **modèles de calcul** : on dit qu'une fonction est **calculable** si elle peut être programmée dans un langage de programmation usuel.

Pour définir la calculabilité d'une fonction, on pourrait utiliser d'autres modèles de calcul comme une machine de Turing *(représentation ci-dessous)*.

> **La machine de Turing (1936)**
> C'est une machine abstraite faite d'un **ruban** infini dont les cases sont composées de caractères d'un alphabet (un nombre de symboles fini), une **tête de lecture** indiquant son état, une **table de transitions** indiquant comment changer d'état.



La ***thèse de Church*** postule que tout problème de calcul fondé sur une procédure algorithmique peut être résolu par une machine de Turing.

> **Qu'est-ce qu'une thèse ? Que peut-on en déduire ?**
> Une thèse est une position, une affirmation sur un sujet donné, que l'on soutient avec des arguments. La **thèse de Church** n'a donc pas été prouvée, mais n'a jamais été contredite.

$\rightarrow$ Dans la suite, on considère qu'une fonction est calculable si elle peut être programmée dans le langage Python (s'il existe un programme donnant le résultat).

2.2 Exemples de problèmes

Un nombre est-t-il premier ?

On appelle ce problème un problème de décision : on doit décider si oui ou non, n est premier. La fonction, si elle existe, qui répond à ce problème, est une fonction f prenant en entrée un entier naturel, et renvoyant en sortie un booléen.

On dit que ce problème est **décidable** si cette fonction f est calculable : c'est-à-dire s'il existe un programme Python permettant de calculer cette fonction f pour tout entier n. Est-ce le cas ?

Une chaîne de caractères est-elle un programme ?

Comment déterminer si ce problème est décidable ? Est-ce le cas *(donner une justification minimale)* ?

2.3 Problème de l'arrêt

Il n'y a pas de programme nous permettant de décider si un programme quelconque va s'arrêter. Alan Turing en fournit le premier la preuve et énonce que **« le problème de l'arrêt est indécidable »**. Tous les problèmes ne peuvent pas être résolus par des algorithmes car toutes les fonctions ne sont pas calculables.

→ **Donnons-en la preuve (par l'absurde) :**

On suppose l'existence d'un tel programme, que l'on nomme `arret`.

`arret(prog, donnee)` prend en paramètres un programme `prog` et une donnée `donnee`. Il renvoie :

- `True` si `prog(donnee)` s'arrête ;
- `False` si `prog(donnee)` ne s'arrête pas.

On définit un deuxième programme `absurde` :

```
def absurde(prog):
    if arret(prog, prog):
        while True
            pass    # boucle infinie
    else:
        return True
```

`absurde` est défini pour un programme quelconque, donc peut être appelé sur lui-même.

On fait l'appel `absurde(absurde)` :

- si `arret(absurde, absurde)` vaut `True` (donc si `absurde(absurde)` s'arrête), alors on rentre dans la boucle infinie ;
- sinon (si `absurde(absurde)` ne s'arrête pas), alors `absurde(absurde)` s'arrête.

On a une contradiction. Notre hypothèse de départ est donc fausse : **il n'existe pas de programme qui puisse déterminer à coup sûr qu'un programme quelconque se termine**.

Le problème de l'arrêt est **indécidable**.

https://trebaul.github.io/nsi