Langages et Programmation

# Décidabilité

## *Les programmes vus comme des données*

La plupart des programmes **attendent en entrée des données *(input)*** pour faire ce qu'on attend d'eux, et produire en sortie un résultat ***(output).***

Exemple :

```
1  def prog(t):
2      m=t[0]
3      for e in t:
4          if e < m:
5              m=e
6      return m
```


Lors de l'appel à la fonction ***prog*** :

Lors de l'appel à la fonction ***prog*** :

- Quel type d'entrée sera attendu ? ……..
- Quelle sortie sera renvoyée ? …….

Ces données d'entrée peuvent être de types très divers :

- Types de base : … …….
- Ou types construits : …….

Cela peut également être des fichiers textes, ou n'importe quel type de fichier, en particulier : il est tout à fait possible d'**avoir en donnée d'entrée un autre programme**

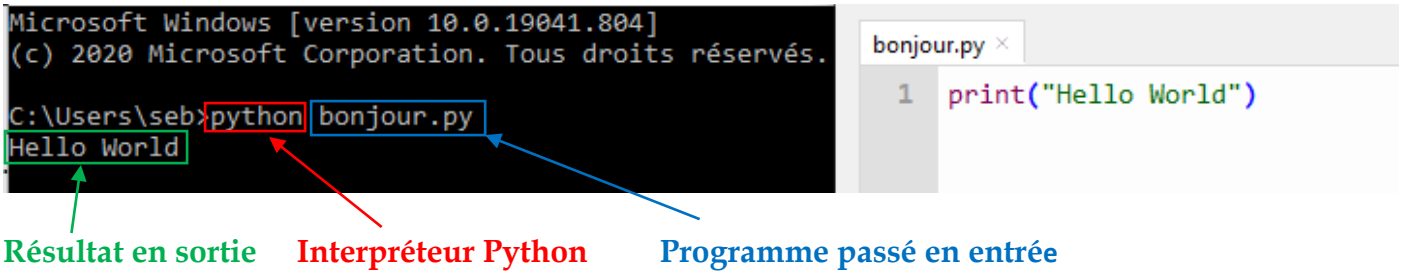

## *Un premier exemple*

Les ordinateurs ne comprennent que le langage machine, c'est-à-dire des 0 et des 1, alors que le programmeur préfère utiliser un langage dit de ''haut niveau'' proche de l'anglais naturel. On doit donc utiliser un programme qui transforme ce langage évolué (Python, C, Java, C++ …) en langage machine. Ce programme est un compilateur ou un interpréteur.

| | | |
|---|---|---|
| ..... | **traduit le programme en entier avant de l'exécuter** | **Ex : C** |
| ..... | **exécute le programme au fur et à mesure qu'il le traduit** | **Ex : Python** |

Dans le cas de Python, voici ce qu'il se passe lorsque l'on exécute le programme.


On le voit encore mieux lorsqu'on appelle l'interpréteur Python directement depuis une fenêtre de commande système, au lieu de passer par Thonny : il faut lui passer le nom du programme en argument :




---

## *D'autres exemples*

---

- Quand on télécharge un logiciel sur Internet : On utilise un programme (un gestionnaire de téléchargement) qui prend un paramètre le logiciel (et donc le programme) à télécharger
- Un système d'exploitation est un ''programme géant'' qui fait tourner (et donc qui exécute) plein de programmes différents

C'est une notion fondamentale qui transparaît de tous ces exemples :

**Tout programme est aussi une donnée**

---

## *Le problème de l'arrêt*

---

Le "cauchemar" d'un programmeur est que son programme, dans certains cas, "tombe" dans une boucle infinie et ne s'arrête jamais. Dans ce cas, le logiciel est incapable de fournir la réponse attendue par l'utilisateur.

- Un programme qui permettrait de tester si un autre programme va finir par s'arrêter, quel que soit le cas traité, serait d'une grande aide pour tous les développeurs du monde !

On se pose alors une question, appelée **problème de l'arrêt** qui pourrait être formulé ainsi :

Existe-il un programme ''universel'' qui pourrait déterminer si un programme se termine ou non en fonction de l'entrée que l'on lui donne ?

La réponse est NON

Voir vidéo : https://www.youtube.com/watch?v=92WHN-pAFCs

Cette propriété découverte par Alonzo Church et Alan Turing en 1937 est énnoncée comme telle :

## Le problème de l'arrêt est indécidable

**Définition :**

On dit qu'un problème est **décidable** s'il peut être résolu par un algorithme.
On dit qu'un problème est **indécidable** dans le cas contraire.

---

## *Calculabilité*

---

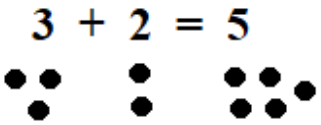

Kurt Gödel


Alan Turing


Alonzo Church

La plupart des fonctions mathématiques peuvent être décrites par une suite de manipulations définies à l'aide de symboles, et visualisées mentalement comme des déplacements de petits cailloux. On parle de fonctions **calculables.**

***« Calcul » vient du latin « calculus » qui signifie « caillou ».***
***Par exemple, on peut matérialiser concrètement les additions, soustractions, multiplications et divisions par des manipulations sur des petits cailloux : ce sont des fonctions calculables.***




**Définition :**

On dit qu'une fonction est **calculable** lorsque l'on peut la programmer grâce à un algorithme :

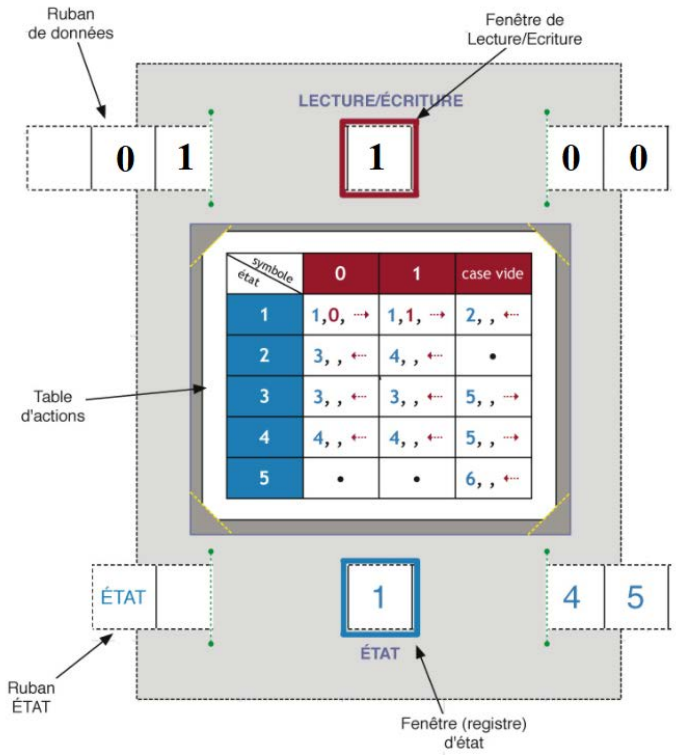
- consistant en **un ensemble fini d'instructions** simples et précises, décrites avec un nombre limité de symboles
- produisant le résultat en **un nombre fini d'étapes**
- pouvant être **suivi par un humain** avec seulement du papier et un crayon
- dont l'exécution ne requiert **aucune ''intelligence''** autre que la compréhension des instructions

Certaines fonctions $f$ très complexes ne vérifient pas les critères ci-dessus. Cela ne signifie pas forcément que l'on ne peut pas trouver la valeur de $f(x)$ mais celle-ci ne s'obtient pas en appliquant un algorithme.

L'exemple le plus connu est celui de la fonction dite du **''castor affairé''** dont la non calculabilité a été démontrée. On peut toutefois trouver certaines valeurs de cette fonction.

**Calculabilité et Décidabilité sont des notions très proches, on s'intéresse aux fonctions dans le cas de la calculabilité alors qu'on s'intéresse aux propriétés dans le cas de la décidabilité.**

## *Machines de Turing*




Le britannique Alan Turing a pensé puis réalisé une machine composée :

- d'un ruban infini divisé en cases, dans lesquelles la machine peut écrire des symboles.
- d'une tête de lecture et d'écriture
- d'une table de transition, dont chaque ligne :
  - est associée à un état
  - spécifie les actions à effectuer quand la machine est dans cet état en fonction du symbole lu par la tête de lecture
- ayant comme actions possibles :
  - Ecrire un symbole (choisi dans un alphabet, 0 ou 1 souvent)
  - Se déplacer d'une case sur la gauche ou sur la droite
  - Changer d'état
- S'arrêtant quand on atteint un état désigné comme « final »

*Activité : Manipuler les machines de Turing avec des Lego*

Le britannique Alan Turing a prouvé en 1937 que **les fonctions calculables étaient exactement les fonctions programmables sur la machine imaginaire qu'il avait conçue**.

Tout programme d'ordinateur, peu importe le langage de programmation, peut être traduit en une machine de Turing. On parle donc de **machine universelle**.

## La calculabilité est donc indépendante du langage de programmation.