> Version pdf

Calculabilité (Bac 🎯)

Programme comme donnée

> **"Programme comme donnée"**

Lorsqu'on exécute un programme Python, il est traité comme une donnée par un autre programme qui est l'interpèteur Python. Par exemple si un fichier `hello.py` contient le code Python `print('Hello World')` on l'exécute depuis la console de commande par défaut du système d'eploitation, en invoquant l'interpréteur Python avec la commande `python` et en lui passant en argument le chemin vers le fichier contenant le programme.

Le programme est donc bien traité comme une donnée par l'interpréteur.

```
>>> python 'hello.py'
Hello World
```

Dans l'architecture de Von Neumann qui est la plus répandue parmi les architectures d'ordinateurs, les programmes et les données sont stockés dans la même mémoire centrale.

Le **système d'exploitation** (OS) est l'intermédiaire entre le matériel et les autres programmes. En particulier, pour démarrer l'exécution d'un programme, l'OS charge comme une donnée le code du programme depuis une mémoire externe dans la mémoire vive (RAM) puis initialise les registres du processeur. Le **processus** d'exécution du programme est alors créé et celui-ci peut à son tour manipuler des données.

Calculabilité et décidabilité

> **"Histoire d'algorithme : des calculus romains à l'Entscheidungsproblem"**

Le mot **calcul** vient du latin *calculus* qui désigne les cailloux utilisés dans les procédures de comptage mécanique pendant l'Antiquité : un caillou pour représenter une unité. Très tôt les

hommes ont imaginé des machines capables de réaliser des calculs : machine d'Anticythère (87 av. J.C.) pour des calculs astronomiques, Pascaline au XVIIème siècle pour les calculs arithmétiques. À la fin du XVIIème siècle, Leibniz imagina le *calculus ratiocinator*, une machine qui pourrait manipuler des symboles pour *"calculer"* des raisonnements logiques. Au XIXème siècle, Charles Babbage conçoit les plans d'une machine capable de réaliser des calculs arithmétiques et logiques et d'exécuter plus généralement des **algorithmes**. C'est l'ancêtre de nos ordinateurs modernes !

> **"Définition d'un algorithme"**
>
> La notion d'**algorithme** généralise la notion de calcul numérique comme séquence finie de règles bien établies. Un **algorithme** est une suite finie et non ambiguë d'opérations bien définies qui s'applique à une entrée et produit une sortie, qui apporte une solution à un problème bien spécifié. Le nom algorithme dérive de celui du mathématicien persan Al-Khwarizmi (780 - 850).

En 1900 les mathématiciens réunis en congrès à Paris discutent des fondements logiques des mathématiques et David Hilbert présente une liste de problèmes à résoudre pour le siècle à venir. Le dixième problème touche aux fondements du calcul et des algorithmes puisqu'il consiste à trouver un procédé déterminant automatiquement si une équation diophantienne a des solutions. En 1928, David Hilbert, formule le problème de la décision ou Entscheidungsproblem.

> **"Entscheidungsproblem (Hilbert 1928, version simplifiée)"**
>
> L'Entscheidungsproblem pose la question suivante : existe-t-il un algorithme qui peut décider si n'importe quel énoncé mathématique bien formulé est vrai ou faux ?
>
> C'est un exemple de **problème de décision** : sur l'ensemble E des énoncés mathématiques, on considère la fonction *f* qui à un énoncé *e* associe `True` ou `False`. Le problème est **décidable** s'il existe un algorithme permettant de calculer *f(e)* pour tout énoncé *e*.

> **"Naissance de l'informatique moderne : des modèles de calcul aux machines**

universelles"

En 1936, Alonzo Church et Alan Turing ont démontré de façon indépendante que l'Entscheidungsproblem est **indécidable** : il ne peut exister d'algorithme capable de déterminer sin'importe quel énoncé mathématique bien formulé est vrai ou faux.

Pour obtenir ce résultat, ils ont développé des modèles théoriques recouvrant ce qu'on désigne intuitivement par **calcul** ou plus généralement **algorithme**. Turing a conçu un modèle fondé sur des machines théoriques et Church sur un langage formel appelé lambda-calcul.

De plus Turing a démontré en 1937 que les fonctions calculables par une **machine de Turing** sont les mêmes que celles calculables par le **lambda-calcul**.

La thèse de Church-Turing est une conjecture (donc non démontrée) qui affirme que la notion intuitive de calcul est entièrement capturée par ces modèles de calcul équivalents de Turing, Church et Kleene.

"Calculabilité"

Une **fonction calculable** est une fonction mathématique f définie sur un ensemble E pour laquelle il existe un algorithme (ou une machine de Turing d'après la thèse de Church-Turing) qui pour tout x appartenant à E, calcule son image $y = f(x)$ par f en un temps fini.

Une fonction **calculable** qui prend ses valeurs dans l'ensemble des booléens est dite **décidable**.

"Machines de Turing et machine universelle"

Une **machine de Turing** permet de modéliser un algorithme. Si on change d'algorithme il faut donc une nouvelle machine de Turing.

Alan Turing a cependant réalisé le tour de force de démontrer qu'on peut construire une **machine de Turing universelle**. Celle-ci peut prendre en entrée sur son ruban la description d'une machine de Turing particulière M ainsi que son entrée E et peut alors simuler l'exécution de M sur En pour écrire en sortie sur son ruban le même résultat que si M s'était appliquée à E.

Dans une **machine de Turing universelle**, un algorithme (= une machine de Turing particulière) est traité comme une *donnée*. Ainsi une seule machine peut exécuter tous les algorithmes (= fonctions calculables) et les algorithmes sont codés dans la machine sous forme de **programmes**.

Les ordinateurs et ordiphones que nous connaissons sont des machines de Turing universelles. Il sont basés sur l'architecture de Von Neumann où **la mémoire contient les programmes et les données** comme le ruban d'une machine de Turing universelle.

On ne peut pas tout calculer : le problème de l'arrêt

"Le problème de l'arrêt est indécidable"

"Énoncé du problème de l'arrêt"

On considère la fonction `arret` telle que pour tout couple d'arguments constitué d'un algorithme `f` et d'une entrée `e`, renvoie :

- `arret(f, e)` renvoie `True` si l'algorithme *f* appliqué à *e* se termine
- `arret(f, e)` renvoie `False` sinon

Existe-t-il un algorithme qui permet de calculer `arret(f, e)` pour tout couple d'arguments `f` et `e`.
Autrement dit, la fonction `arret` est-elle **calculable** et plus précisément **décidable** puisqu'elle est à valeurs booléennes ?

Dans son article *"On Computable Numbers, with an Application to the Entscheidungsproblem"* publié en janvier 1937, Alan Turing a démontré que la fonction `arret` est indécidable. Il en a déduit que l'Entscheidungsproblem est indécidable.

"Importance du résultat"

Le fait que le **problème de l'arrêt est indécidable** fixe une *limite dure* et *universelle* au pouvoir des algorithmes : on ne peut pas tout calculer !