

# Introduction à la Programmation des Algorithmes

## 1.2. Introduction – Langages de programmation

François Fleuret

<https://fleuret.org/11x001/>



**UNIVERSITÉ  
DE GENÈVE**

“D’une manière similaire à une langue naturelle, un langage de programmation est composé d’un alphabet, d’un vocabulaire, de règles de grammaire, de significations, mais aussi d’un environnement de traduction censé rendre sa syntaxe compréhensible par la machine.”

(Wikipédia)

## Langages naturels

- Français.
- Anglais-US.
- Langage SMS.

## Langages formels

- Description de parties d'échecs.
- Notation mathématique.
- Langage HTML.
- Langages de programmation.

- **Lexique:** ensemble des symboles et termes acceptables (Vocabulaire).
- **Syntaxe:** ensemble des règles définissant la construction cohérentes de programmes (Grammaire).
- **Sémantique:** ensemble des règles permettant d'associer un sens à une expression (Signification).

Un ordinateur manipule des informations **sans référence au sens qu'elles peuvent avoir.**

Par exemple:

- calculer le résultat d'un vote sans information sur le but du vote,
- réaliser un dessin/une figure sans information sur sa signification,
- rechercher des mots dans un texte sans en comprendre la langue.

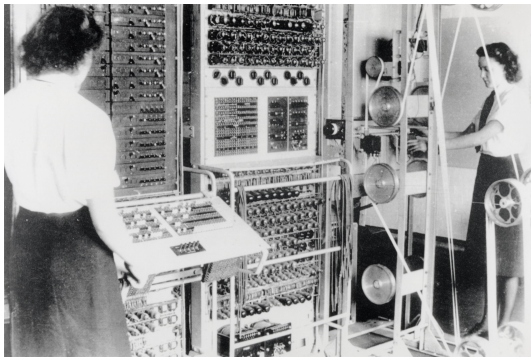
Un ordinateur manipule des informations **sans référence au sens qu'elles peuvent avoir.**

Par exemple:

- calculer le résultat d'un vote sans information sur le but du vote,
- réaliser un dessin/une figure sans information sur sa signification,
- rechercher des mots dans un texte sans en comprendre la langue.

Attribuer un sens à des données est complexe philosophiquement et techniquement. L'IA moderne répond en partie à cette question.

Les premiers ordinateurs se “programmaient” en modifiant leur structure physique: ajouts de cables, interrupteurs.



Un ordinateur cryptographique Colossus Mark 2 (1943)

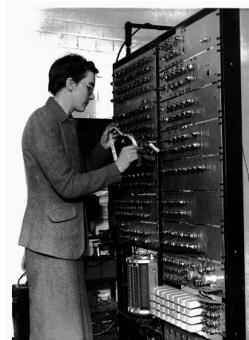
## Langage machine

- Langage natif du microprocesseur.
- Trop rébarbatif pour un humain.
- Différent pour chaque famille de processeur.
- Instructions représentées par un code en hexadécimal (opcode):
  - Addition: 0x05,
  - Multiplication: 0xF6,
  - etc.



## Assembleur

- Traduisible directement en langage machine.
- Instructions représentées par quelques lettres.
- Aussi rébarbatif mais plus lisible.
- Dépend fortement du processeur.



Kathleen Booth

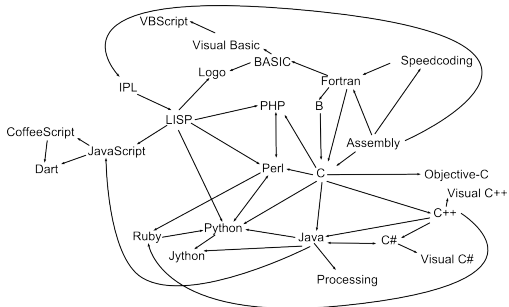
Calculer le PGCD de deux entiers en C:

```
int gcd(int a, int b) {  
    while (a != b) {  
        if (a > b)  
            a = a - b;  
        else  
            b = b - a;  
    }  
  
    return a;  
}
```

Calculer le PGCD de deux entiers en assembleur ARM:

```
gcd    CMP    r0, r1
        BEQ    end
        BLT    less
        SUBS   r0, r0, r1
        B     gcd
less   SUBS   r1, r1, r0
        B     gcd
end
```

## Il existe un très grand nombre de langages de programmation



<https://thomasinterestingblog.wordpress.com/2011/11/26/the-family-tree-of-programming-languages/>

## Langages de bas niveau

Descriptions et concepts proches du matériel.  
Octets, voltages, pixels, etc.

## Langages de haut niveau

Descriptions et concepts proches du problème à résoudre.  
Systèmes d'équations, images, bases de données, fenêtres, pages web.

Abstraction



**Logiciel**

**Matériel**

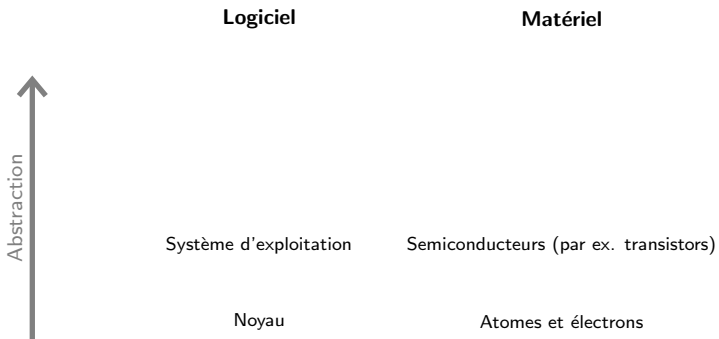
↑  
Abstraction

**Logiciel**

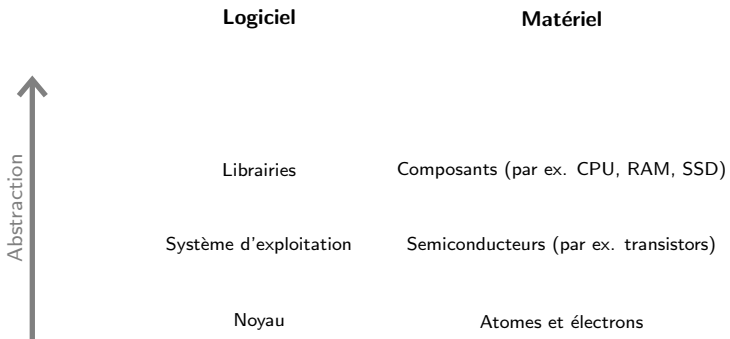
**Matériel**

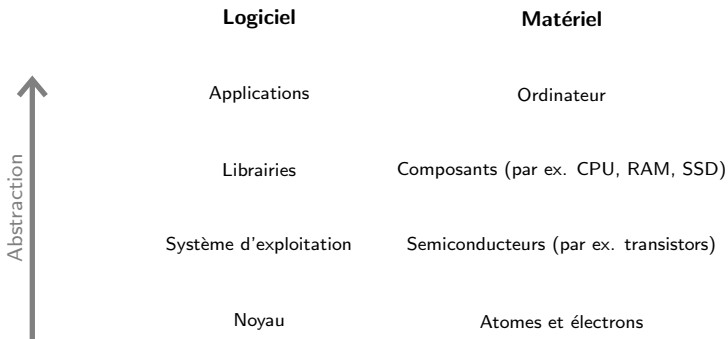
Noyau

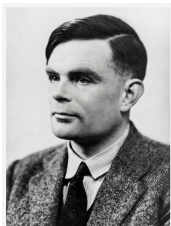
Atomes et électrons





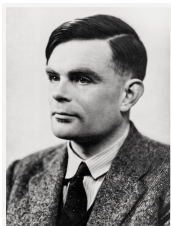






Alan Turing

Alan Turing a proposé en 1936 un modèle mathématique d'un ordinateur, communément appelé "machine de Turing". Ce modèle est considéré comme suffisant pour réaliser n'importe quel calcul utile.



Alan Turing

Alan Turing a proposé en 1936 un modèle mathématique d'un ordinateur, communément appelé "machine de Turing". Ce modèle est considéré comme suffisant pour réaliser n'importe quel calcul utile.

La vaste majorité des langages de programmation sont dits *Turing complete*, c'est à dire qu'ils permettent de programmer une machine de Turing, et par conséquent permettent de programmer n'importe quel calcul.

Un programme peut parfois mal fonctionner, parce qu'il traite des données qui n'ont pas une propriété attendue (par ex. nombre non-nul) ou parce qu'il contient une erreur de conception.

Un programme peut parfois mal fonctionner, parce qu'il traite des données qui n'ont pas une propriété attendue (par ex. nombre non-nul) ou parce qu'il contient une erreur de conception.

Il est extrêmement difficile de faire des logiciels pour lesquels on a une assurance formelle qu'ils ne feront aucune erreur. Si cela est possible c'est au prix de contraintes énormes pour les programmeurs.

Un programme peut parfois mal fonctionner, parce qu'il traite des données qui n'ont pas une propriété attendue (par ex. nombre non-nul) ou parce qu'il contient une erreur de conception.

Il est extrêmement difficile de faire des logiciels pour lesquels on a une assurance formelle qu'ils ne feront aucune erreur. Si cela est possible c'est au prix de contraintes énormes pour les programmeurs.

Il est en particulier impossible d'écrire un programme capable de déterminer automatiquement si un programme se termine (*halting problem*).

**Tout programme écrit par un humain doit être traduit en langage machine pour être compris et exécuté par le processeur.**

On parle de programme:

- **Compilé:** lorsque la traduction a lieu avant l'exécution. Un logiciel nommé **compilateur** fait cette traduction.
- **Interprété:** lorsque la traduction a lieu pendant l'exécution. Un logiciel nommé **interpréteur** fait cette traduction.

*A priori* tous les langages peuvent être compilés ou interprétés, mais la plupart sont conçus comme l'un ou l'autre.



**Compilé**

---

Architecture choisie à l'avance  
Des vérifications avant l'exécution  
Plus facile à optimiser  
Programmation plus rigide  
Gros fichiers exécutables

---

**Interprété**

---

Architecture choisie à l'exécution  
Surprises à l'exécution  
Plus difficile à optimiser  
Programmation plus flexible  
Petits exécutables

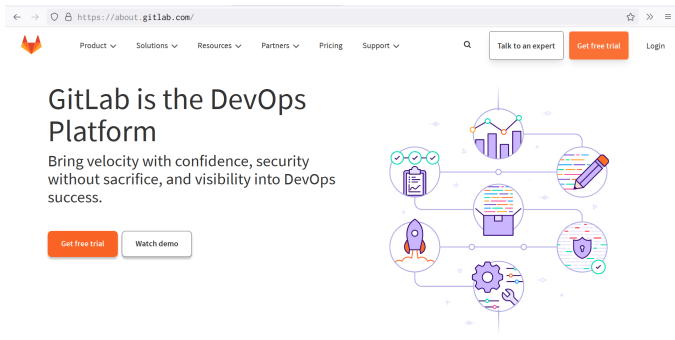
---

## Nombres de lignes de code dans le noyau Linux (01.01.2020).

Language	files	blank	comment	code
C	27961	2761653	2292505	14061980
C/C++ Header	19875	531782	956091	4300989
reStructuredText	2153	101820	53725	270579
Assembly	1320	46971	101470	230117
JSON	273	0	0	161955
Bourne Shell	577	13299	9690	52817
make	2531	9485	10644	41766
SVG	58	117	1364	36216
Perl	59	6021	4422	30642
Python	118	4987	4588	26256
YAML	325	5205	1449	25529
yacc	9	697	359	4810
PO File	5	791	918	3077
lex	8	326	300	2015
C++	10	320	129	1933
Bourne Again Shell	51	356	297	1765
awk	10	140	116	1060
Glade	1	58	0	603
NAnt script	2	146	0	551
Cucumber	1	28	50	174
Windows Module Definition	2	15	0	109
m4	1	15	1	95
CSS	1	28	29	80
XSLT	5	13	26	61
vim script	1	3	12	27
Ruby	1	4	0	25
INI	1	1	0	6
sed	1	2	5	5
<b>Total</b>	<b>55360</b>	<b>3484283</b>	<b>3438190</b>	<b>19255242</b>

Le développement de logiciels complexes de manière collaborative, en particulier par des équipes distribuées géographiquement, demande des outils adaptés qui ne sont pas propre à un langage de programmation.

Le plus important de ces outils est git, pour lequel existe de nombreuses interfaces web et sites d'hébergement.



The image shows a screenshot of the GitLab website homepage. The browser address bar displays "https://about.gitlab.com/". The navigation menu includes "Product", "Solutions", "Resources", "Partners", "Pricing", and "Support". There are buttons for "Talk to an expert", "Get free trial", and "Login". The main heading reads "GitLab is the DevOps Platform". Below this, the text states: "Bring velocity with confidence, security without sacrifice, and visibility into DevOps success." There are two buttons: "Get free trial" and "Watch demo". On the right side, there is a diagram illustrating the DevOps lifecycle with icons for monitoring, deployment, security, and collaboration.

# Langage C et Python

## Langage C



- Créé en 1972 par Dennis Richie en même temps qu'Unix.
- Simple et rapide.
- Primitif et proche de la machine.
- Aucune gestion de la mémoire et des erreurs.
- Utilisé par ex. pour le noyau Linux, autres langages, pilotes de périphériques.

**Langage utilisé pour développer des programmes quand rien ne pré-existe ou quand le contrôle du matériel doit être total. Ne cache aucun mécanisme au programmeur.**

## Le C est un langage compilé

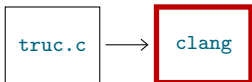
- Le programmeur écrit le fichier source `truc.c`,



`truc.c`

## Le C est un langage compilé

- Le programmeur écrit le fichier source `truc.c`,
- le fichier exécutable du compilateur `clang` prend ce fichier source en entrée



## Le C est un langage compilé

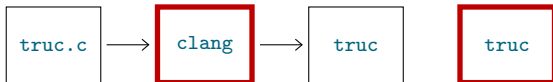
- Le programmeur écrit le fichier source `truc.c`,
- le fichier exécutable du compilateur `clang` prend ce fichier source en entrée et produit un fichier exécutable `truc`,





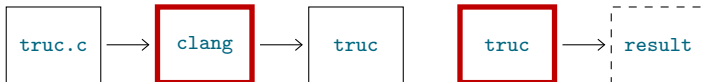
## Le C est un langage compilé

- Le programmeur écrit le fichier source `truc.c`,
- le fichier exécutable du compilateur `clang` prend ce fichier source en entrée et produit un fichier exécutable `truc`,
- ce fichier peut être exécuté



## Le C est un langage compilé

- Le programmeur écrit le fichier source `truc.c`,
- le fichier exécutable du compilateur `clang` prend ce fichier source en entrée et produit un fichier exécutable `truc`,
- ce fichier peut être exécuté et produit le résultat.



En C, on écrit un **code source**

```
#include <stdio.h>

int main(void) {
    int a;
    a = 1;
    while (a < 100) {
        printf("%d\n", a);
        a = a * 3;
    }
    return 0;
}
```

que l'on sauve dans un fichier (ici `truc.c`), que l'on **compile** en un fichier **exécutable** (ici `truc`) que l'on peut faire fonctionner

```
~ clang truc.c -o truc
~ ./truc
1
3
9
27
81
```

On peut indiquer au compilateur de s'arrêter à la traduction en assembleur.

```
~ clang -S truc.c
```

il produit alors un fichier truc.s

```
.text
.file    "truc.c"
.globl  main                    # -- Begin function main
.p2align    4, 0x90
.type    main,@function

main:                                # @main
.cfi_startproc
# %bb.0:                               # %entry
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    movl    $0, -4(%rbp)
    movl    $1, -8(%rbp)

.LBB0_1:                               # %while.cond
                                        # =>This Inner Loop Header: Depth=1
    cmpl    $100, -8(%rbp)
    jge     .LBB0_3
```

Nous allons étudier une partie du langage C afin de bien comprendre les mécanismes de bas niveau qui ont lieu dans un ordinateur.

C'est un langage primitif qui demande un grand soin de programmation pour réaliser des programmes longs et sans erreur.

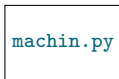


- Créé en 1991 par Guido van Rossum.
- Complet et lent.
- Fournit des mécanismes orientés objets et fonctionnels.
- Gestion de la mémoire et des erreurs.
- Utilisé par ex. pour des utilitaires, applications web, IA.

**Langage utilisé en pratique pour le développement d'applications. Repose sur des mécanismes (cachés) complexes. Offre un immense choix de bibliothèques.**

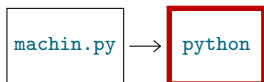
## Python est un langage interprété

- Le programmeur écrit le fichier source `machin.py`,

A rectangular box with a thin black border containing the text `machin.py` in a blue monospace font.

## Python est un langage interprété

- Le programmeur écrit le fichier source `machin.py`,
- le fichier exécutable du compilateur `python` prend ce fichier source en entrée





## Python est un langage interprété

- Le programmeur écrit le fichier source `machin.py`,
- le fichier exécutable du compilateur `python` prend ce fichier source en entrée et produit le résultat.



On écrit un **code source**

```
a = 1
while a < 100:
    print(a)
    a = a * 3
```

que l'on sauve dans un fichier (ici [machin.py](#)), que l'on donne comme entrée à un interpréteur Python que l'on **exécute**.

```
~ python ./machin.py
1
3
9
27
81
```

Fin