COURS D'INITIATION · INFORMATIQUE

Les langages bas niveau & haut niveau

Comprendre comment on parle aux ordinateurs : du binaire le plus brut jusqu'aux langages modernes. Un guide complet, illustré et pas à pas, pensé pour les grands débutants.

Niveau : débutant · Aucun prérequis

01000010 01101001 01101110 01100001 01101001 01110010 01100101

Support de cours pédagogique · Format A4

Sommaire

01 Avant tout : qu'est-ce qu'un langage de programmation ?
Parler à une machine · pourquoi tant de langages existent

02 Comment l'ordinateur « réfléchit »
Le binaire · le processeur · la notion d'instruction

03 La grande idée : le niveau d'abstraction
L'échelle qui sépare le bas niveau du haut niveau

04 Les langages bas niveau
Langage machine · assembleur · avantages et limites

05 Les langages haut niveau
Caractéristiques · exemples · le cas particulier du C

06 Du code au programme : compilation vs interprétation
Comment votre texte devient une application qui tourne

07 Comparaison côte à côte
Tableau récapitulatif · le même programme dans 4 langages

08 Par où commencer ? Conseils pour le débutant
Quel langage choisir · outils · bonnes habitudes

09 Glossaire des mots à connaître

10 Quiz & exercices corrigés

CHAPITRE 1

Qu'est-ce qu'un langage de programmation ?

Avant de distinguer « bas niveau » et « haut niveau », posons les bases : pourquoi a-t-on inventé des langages pour parler aux ordinateurs ?

Un ordinateur est une machine extraordinairement rapide mais totalement dépourvue d'intuition. Il ne « comprend » rien au sens humain : il exécute, une par une, des instructions très précises. Un **langage de programmation** est l'ensemble de mots, de symboles et de règles qui nous permet d'écrire ces instructions de manière organisée.

DÉFINITION

Un **langage de programmation** est un langage artificiel servant à donner des ordres à un ordinateur. Comme une langue humaine, il possède un *vocabulaire* (les mots-clés) et une *grammaire* (la syntaxe) qu'il faut respecter pour être « compris ».

ANALOGIE

Imaginez que vous deviez expliquer à quelqu'un comment faire un café, mais que cette personne ne sait *rien* faire toute seule. Vous ne pouvez pas dire « fais-moi un café ». Vous devez préciser chaque geste : « prends la tasse », « remplis le réservoir d'eau », « appuie sur le bouton »… Programmer, c'est exactement cela : décomposer une tâche en gestes minuscules et sans ambiguïté.

Pourquoi existe-t-il autant de langages ?

Il existe des centaines de langages de programmation. Cette diversité n'est pas un caprice : chaque langage a été créé pour répondre à un besoin particulier — la vitesse, la simplicité, la sécurité, le calcul scientifique, les sites web, les jeux vidéo… Tous ne sont pas au même « niveau ». Et c'est précisément là qu'apparaît la distinction centrale de ce cours.

À RETENIR

Il n'existe pas de « meilleur » langage dans l'absolu. Il existe des langages plus ou moins **adaptés à un objectif**, et plus ou moins **proches de la machine**. Ce deuxième critère — la proximité avec la machine — est ce qui sépare les langages bas niveau des langages haut niveau.

CHAPITRE 2

Comment l'ordinateur « réfléchit »

Pour comprendre ce que veut dire « proche de la machine », il faut savoir ce que la machine sait faire, tout au fond.

Tout n'est que 0 et 1 : le binaire

Au cœur d'un ordinateur, il n'y a que de l'électricité : le courant passe, ou il ne passe pas. On représente ces deux états par deux chiffres : `0` (pas de courant) et `1` (courant). C'est le **système binaire**. Chaque **0** ou **1** s'appelle un *bit*.

> **DÉFINITION**
>
> Le **binaire** est un système qui n'utilise que deux chiffres, 0 et 1. Toutes les informations d'un ordinateur — textes, images, sons, programmes — sont au final stockées sous forme de longues suites de 0 et de 1. Un groupe de 8 bits forme un **octet** (par exemple `01000001` représente la lettre « A »).

Le processeur : le chef d'orchestre

Le **processeur** (ou **CPU**, pour *Central Processing Unit*) est le composant qui exécute les instructions. Mais attention : il ne comprend qu'un seul « langage », un répertoire très restreint d'opérations élémentaires codées en binaire, par exemple : « additionne ces deux nombres », « copie cette valeur ici », « compare ces deux valeurs ».

> **ANALOGIE**
>
> Le processeur est comme un employé d'usine ultra-rapide mais qui ne connaît qu'une poignée de gestes simples. Donnez-lui un geste qu'il connaît et il l'exécute des milliards de fois par seconde. Mais il est incapable de comprendre une consigne complexe formulée en langage humain : il faut la lui traduire en gestes élémentaires.

La notion d'instruction

Programmer revient donc à fournir au processeur une liste d'**instructions** qu'il sait exécuter. Le grand problème : ces instructions, dans leur forme la plus brute, sont des suites de 0 et de 1, illisibles pour un humain. Toute l'histoire des langages de programmation est une longue quête pour s'éloigner de ce binaire et écrire d'une manière plus naturelle, tout en restant traduisible vers la machine.

> **LE FIL ROUGE DU COURS**
>
> Plus un langage est **proche du binaire** et du fonctionnement réel du processeur, plus il est dit de **bas niveau**. Plus il est **proche du langage humain** et masque les détails de la machine, plus il est dit de **haut niveau**.

CHAPITRE 3

La grande idée : le niveau d'abstraction

« Bas » et « haut » niveau ne sont pas un jugement de valeur. Ils décrivent une distance : la distance entre votre code et le matériel.

Le mot-clé : abstraction

DÉFINITION

L'**abstraction**, c'est le fait de cacher les détails compliqués derrière quelque chose de simple à utiliser. Plus un langage offre d'abstraction, plus il vous épargne les détails techniques de la machine — et plus il est de « haut niveau ».

ANALOGIE : LA VOITURE

Conduire une voiture moderne, c'est du **haut niveau** : vous tournez le volant et appuyez sur une pédale, sans rien savoir des pistons et de l'injection. Réparer le moteur pièce par pièce, c'est du **bas niveau** : vous touchez directement la mécanique. Les deux servent à « faire avancer la voiture », mais à des distances très différentes du fonctionnement réel.

Le spectre, du plus bas au plus haut

Il ne faut pas imaginer deux camps séparés, mais plutôt une échelle continue :

◀ BAS NIVEAU (proche de la machine) **HAUT NIVEAU (proche de l'humain) ▶**

Une autre façon de voir : les couches

On peut aussi représenter un ordinateur comme un empilement de couches. En bas, le matériel ; en haut, vous. Chaque couche cache la complexité de celle du dessous.

Vous (l'humain)
une idée : « je veux afficher un message »

Langage haut niveau — ex. Python
code lisible, proche du français : `print("Bonjour")`

Langage bas niveau — assembleur
instructions élémentaires du processeur

Langage machine — binaire
suites de 0 et de 1

Le matériel — processeur & transistors
du courant qui passe ou non

À RETENIR

« Bas » ne veut pas dire « mauvais » et « haut » ne veut pas dire « meilleur ». Ce sont des positions sur une échelle de proximité avec le matériel. Chaque position a ses forces et ses faiblesses, que nous allons détailler.

CHAPITRE 4

Les langages bas niveau

Ce sont les langages les plus proches du processeur. Ils offrent un contrôle total… au prix d'une grande difficulté.

4.1 — Le langage machine

C'est le seul langage que le processeur comprend *réellement* et directement. Il est constitué uniquement de suites de 0 et de 1 (parfois écrites en notation hexadécimale pour gagner de la place). Personne, aujourd'hui, ne programme directement en langage machine : ce serait extrêmement lent et source d'erreurs.

UNE INSTRUCTION EN LANGAGE MACHINE (EXEMPLE)

```
10110000 01100001
```

Cette suite pourrait signifier, pour un certain processeur, « place la valeur 97 dans une case mémoire ». Imaginez écrire un programme entier comme cela…

> **DÉFINITION**
>
> Le **langage machine** (ou *code machine*) est l'ensemble des instructions binaires directement exécutables par un processeur. C'est le niveau le plus bas qui soit.

4.2 — L'assembleur

Pour rendre le langage machine un peu plus supportable, on a inventé l'**assembleur** (ou *assembly*). L'idée est simple : remplacer les suites de 0 et de 1 par des mots-codes courts, faciles à mémoriser, appelés **mnémoniques**. Chaque mnémonique correspond presque exactement à une instruction machine.

PROGRAMME EN ASSEMBLEUR (ADDITIONNER 5 ET 3)

```
; on charge le nombre 5 dans un registre
MOV AX, 5
; on ajoute 3 au contenu de ce registre
ADD AX, 3
; AX contient maintenant 8
```

C'est plus lisible que le binaire — `MOV` veut dire « *move* » (déplacer/charger), `ADD` veut dire « ajouter » — mais on reste très proche de la machine : on manipule directement des **registres** (de minuscules cases de mémoire interne au processeur), et chaque ligne ne fait qu'une opération minuscule.

DÉFINITION

L'**assembleur** est un langage bas niveau qui remplace les codes binaires par des mots-codes lisibles (les mnémoniques). Il reste spécifique à une famille de processeurs : un programme assembleur écrit pour un type de processeur ne fonctionne pas sur un autre.

4.3 — Avantages et inconvénients du bas niveau

Avantages 👍	Inconvénients 👎
Contrôle total et précis du matériel et de la mémoire.	Très difficile à écrire, à lire et à corriger.
Programmes très rapides et très légers.	Extrêmement long : il faut beaucoup de lignes pour peu de résultat.
Indispensable pour certaines tâches critiques (pilotes, systèmes embarqués).	Non portable : lié à un type de processeur précis.
Permet de comprendre en profondeur le fonctionnement de la machine.	Risque d'erreurs élevé ; peu adapté aux débutants.

ATTENTION

On n'apprend généralement **pas** à programmer en commençant par l'assembleur. C'est un domaine de spécialistes (sécurité informatique, micro-contrôleurs, optimisation extrême). En tant que débutant, il vous suffit de *comprendre* ce qu'est le bas niveau, sans devoir le pratiquer.

CHAPITRE 5

Les langages haut niveau

Ce sont les langages que l'on utilise au quotidien pour créer des sites, des applications, des jeux. Ils sont conçus pour les humains.

5.1 — Qu'est-ce qui définit un langage haut niveau ?

Un langage haut niveau cache les détails de la machine. Vous ne gérez plus les registres ni les adresses mémoire à la main : le langage s'en occupe pour vous. La syntaxe ressemble à de l'anglais simplifié et aux mathématiques, ce qui la rend bien plus facile à lire.

LE MÊME CALCUL (5 + 3) EN PYTHON — UN LANGAGE HAUT NIVEAU

```
# on additionne et on affiche le résultat
resultat = 5 + 3
print(resultat)   # affiche : 8
```

Comparez ces 2 lignes avec le programme en assembleur du chapitre précédent : c'est plus court, plus clair, et cela fonctionnera sur n'importe quel ordinateur capable d'exécuter du Python.

> **DÉFINITION**
>
> Un **langage haut niveau** est un langage proche de la pensée humaine, qui masque le fonctionnement interne de la machine. Il est généralement plus simple à apprendre, plus rapide à écrire, et **portable** (le même code marche sur différentes machines).

5.2 — Quelques langages haut niveau courants

- **Python · simple, polyvalent**
- **JavaScript · le web**
- **Java · applications, Android**
- **C# · jeux, logiciels Windows**
- **Ruby · sites web**
- **Swift · apps iPhone**
- **PHP · sites web côté serveur**
- **Kotlin · Android moderne**

5.3 — Avantages et inconvénients du haut niveau

Avantages 👍	Inconvénients 👎
Facile à apprendre et à lire, même pour un débutant.	Légèrement moins rapide que le bas niveau (souvent imperceptible).
On écrit beaucoup moins de lignes pour le même résultat.	Moins de contrôle fin sur le matériel et la mémoire.
Portable : le même code fonctionne sur plusieurs machines.	Cache des détails utiles à connaître pour aller plus loin.
Énormément d'outils et de bibliothèques toutes prêtes.	Consomme un peu plus de mémoire.

5.4 — Le cas particulier du C : « niveau intermédiaire »

BON À SAVOIR

Certains langages, comme le **C**, se situent *entre les deux*. On les qualifie parfois de « niveau intermédiaire » : ils sont plus lisibles que l'assembleur, mais permettent encore de manipuler la mémoire de près. Le C est d'ailleurs le langage avec lequel sont écrits beaucoup de systèmes d'exploitation. Cela confirme que la frontière bas/haut niveau est un **dégradé**, pas un mur.

CHAPITRE 6

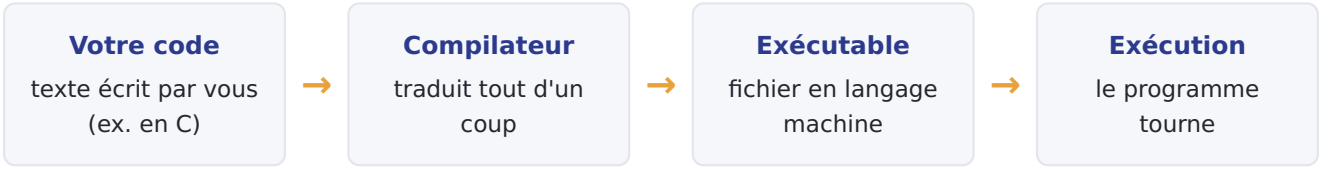
Du code au programme : compilation vs interprétation

Le processeur ne comprend que le binaire. Alors comment votre code haut niveau finit-il par tourner ? Il faut le traduire.

Quel que soit le langage haut niveau utilisé, il faut toujours, à un moment, redescendre vers le langage machine. Cette traduction se fait de deux grandes manières.

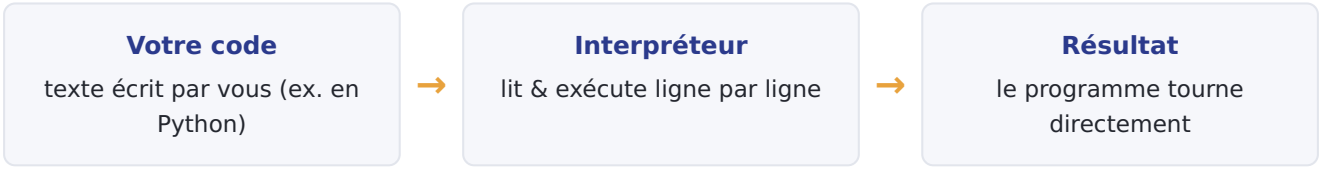
6.1 — La compilation

Un **compilateur** traduit l'intégralité de votre code en langage machine *une fois pour toutes*, avant l'exécution. Le résultat est un fichier exécutable que la machine lance directement. C'est rapide à l'exécution, mais il faut « recompiler » à chaque modification.



6.2 — L'interprétation

Un **interpréteur** lit et traduit votre code *ligne par ligne, au moment où on le lance*. Pas de fichier exécutable à fabriquer à l'avance : c'est pratique pour tester rapidement, mais légèrement plus lent. Python et JavaScript fonctionnent ainsi.



DÉFINITIONS

Compilateur : traduit tout le code en machine *avant* de l'exécuter, produisant un fichier exécutable.
Interpréteur : traduit et exécute le code *au fil de l'eau*, sans créer de fichier exécutable séparé.

À RETENIR

Le binaire n'est jamais « optionnel ». Même avec un langage très haut niveau, votre ordinateur finit toujours par exécuter du langage machine. La compilation et l'interprétation sont simplement deux façons de faire cette traduction.

CHAPITRE 7

Comparaison côte à côte

Récapitulons tout dans un tableau, puis voyons le même mini-programme exprimé à différents niveaux.

7.1 — Tableau récapitulatif

Critère	Bas niveau	Haut niveau
Proximité avec la machine	Très proche	Éloigné (abstrait)
Lisibilité pour l'humain	**Faible**	**Élevée**
Facilité d'apprentissage	**Difficile**	**Facile**
Vitesse d'exécution	**Maximale**	Très bonne
Portabilité (multi-machines)	**Non**	**Oui**
Contrôle du matériel	**Total**	Limité
Nombre de lignes à écrire	Beaucoup	Peu
Exemples	Langage machine, assembleur	Python, Java, JavaScript…
Idéal pour…	Pilotes, systèmes embarqués, sécurité	Sites, applis, jeux, débuter

7.2 — Le même programme à différents niveaux

Objectif : **additionner 5 et 3**. Observez comme on s'éloigne du binaire en montant en niveau.

NIVEAU LE PLUS BAS — LANGAGE MACHINE (BINAIRE, ILLUSTRATIF)

```
10110000 00000101
00000100 00000011
```

BAS NIVEAU — ASSEMBLEUR

```
MOV AX, 5
ADD AX, 3
```

NIVEAU INTERMÉDIAIRE — LANGAGE C

```c
int resultat = 5 + 3;
printf("%d", resultat);
```

HAUT NIVEAU — PYTHON

```python
print(5 + 3)
```

LE CONSTAT

Plus on monte en niveau, plus le code se rapproche de ce qu'un humain écrirait naturellement, et plus il est court. C'est tout l'intérêt du haut niveau : **en dire plus avec moins**.

CHAPITRE 8

Par où commencer ? Conseils pour débuter

Maintenant que la théorie est claire, voici comment passer concrètement à la pratique sans se décourager.

8.1 — Quel langage choisir en premier ?

Pour un grand débutant, le consensus est très large : **commencez par un langage haut niveau**. Vous vous concentrerez sur la logique de programmation au lieu de vous battre avec la machine.

Votre but	Langage conseillé pour débuter
Apprendre les bases / polyvalence	**Python** — le plus recommandé pour démarrer
Faire des sites web interactifs	**JavaScript** (avec HTML & CSS)
Développer des applications Android	**Kotlin** ou Java
Comprendre la machine en profondeur	**C** (un peu plus exigeant)

> **RECOMMANDATION**
>
> Si vous hésitez : choisissez **Python**. Sa syntaxe est épurée, proche de l'anglais courant, et il est utilisé partout (science des données, web, automatisation, intelligence artificielle). C'est un excellent tremplin vers tous les autres langages.

8.2 — De quels outils avez-vous besoin ?

- **Un éditeur de code** : un logiciel pour écrire votre code confortablement (par exemple Visual Studio Code, gratuit).
- **L'interpréteur ou le compilateur** du langage choisi (pour Python, il s'agit simplement d'installer Python).
- **De la curiosité et de la patience** : les erreurs font partie de l'apprentissage, ce ne sont pas des échecs.

8.3 — Bonnes habitudes dès le départ

1. **Pratiquez un peu chaque jour** plutôt que beaucoup d'un coup. La régularité bat l'intensité.
2. **Écrivez de petits programmes** : une calculatrice, un jeu du nombre mystère, une liste de courses.

3. **Lisez vos messages d'erreur** : ils indiquent presque toujours où est le problème.
4. **Commentez votre code** pour expliquer ce qu'il fait (les lignes commençant par `#` en Python).
5. **Ne cherchez pas la perfection.** Un programme qui marche, même imparfait, vaut mieux qu'un programme parfait jamais terminé.

ANALOGIE FINALE

Apprendre à programmer, c'est comme apprendre une langue vivante : on commence par des phrases simples, on fait des fautes, puis on construit des choses de plus en plus complexes. Vous n'avez pas besoin de comprendre le bas niveau pour écrire vos premiers programmes — il vous suffit de savoir qu'il existe, tout en bas de l'échelle.

CHAPITRE 9

Glossaire des mots à connaître

Les termes essentiels rencontrés dans ce cours, expliqués simplement.

Terme	Définition simple
Bit	La plus petite information possible : un 0 ou un 1.
Octet	Un groupe de 8 bits (par ex. `01000001`).
Binaire	Système d'écriture qui n'utilise que 0 et 1.
Processeur (CPU)	Le composant qui exécute les instructions de l'ordinateur.
Registre	Une minuscule case de mémoire interne au processeur.
Instruction	Un ordre élémentaire donné au processeur.
Langage machine	Les instructions en binaire, directement comprises par le processeur.
Assembleur	Langage bas niveau utilisant des mots-codes au lieu du binaire.
Mnémonique	Mot-code court de l'assembleur (ex. `MOV`, `ADD`).
Bas niveau	Langage proche de la machine, puissant mais difficile.
Haut niveau	Langage proche de l'humain, simple et portable.
Abstraction	Le fait de cacher la complexité derrière quelque chose de simple.
Portabilité	La capacité d'un code à fonctionner sur plusieurs machines différentes.
Compilateur	Traduit tout le code en machine avant l'exécution.
Interpréteur	Traduit et exécute le code ligne par ligne.
Syntaxe	Les règles d'écriture d'un langage (sa « grammaire »).
Exécutable	Fichier que la machine peut lancer directement.

CHAPITRE 10

Quiz & exercices corrigés

Testez votre compréhension. Essayez de répondre avant de lire le corrigé !

Question 1. Quels sont les deux seuls chiffres utilisés par le système binaire ?

Réponse : Le 0 et le 1. Chaque chiffre s'appelle un *bit*.

Question 2. Un langage « proche du langage humain et facile à lire » est de quel niveau ?

Réponse : De *haut niveau* (par exemple Python). Le bas niveau, lui, est proche de la machine.

Question 3. L'assembleur est-il un langage bas niveau ou haut niveau ?

Réponse : Bas niveau. Il remplace le binaire par des mots-codes (mnémoniques) mais reste très proche du processeur.

Question 4. Quelle est la différence entre un compilateur et un interpréteur ?

Réponse : Le compilateur traduit tout le code en une fois *avant* l'exécution (et crée un fichier exécutable) ; l'interpréteur traduit et exécute le code *ligne par ligne* au moment où on le lance.

Question 5. Pourquoi conseille-t-on à un débutant de commencer par un langage haut niveau ?

Réponse : Parce qu'il est plus simple à lire et à écrire : le débutant se concentre sur la logique de programmation au lieu de gérer les détails techniques de la machine.

Question 6. « Bas niveau » signifie-t-il « mauvais » ?

Réponse : Non ! « Bas » et « haut » indiquent seulement la distance avec le matériel. Le bas niveau est même indispensable pour certaines tâches (pilotes, systèmes embarqués).

Exercice. Classez ces éléments du plus bas au plus haut niveau : Python, langage machine, assembleur, C.

Corrigé : 1) Langage machine (binaire) → 2) Assembleur → 3) C (intermédiaire) → 4) Python (haut niveau).

Félicitations ! Vous connaissez maintenant la différence fondamentale entre les langages bas niveau et haut niveau, et vous avez toutes les bases pour choisir votre premier langage et commencer à programmer.

Bonne route dans le monde du code. 🚀