

# Module 1 — Introduction a Claude Code

Jour 1 — L'outil agentique

The Bearded Bear

Février 2026



## Table des matières

<b>Module 1 : Introduction a Claude Code</b>	<b>3</b>
Objectifs . . . . .	3
1. Qu'est-ce que Claude Code? . . . . .	3
1.1 Definition . . . . .	3
1.2 Approche agentique . . . . .	3
2. Comparaison avec les autres outils . . . . .	4
2.1 Tableau comparatif . . . . .	4
2.2 Quand utiliser quoi? . . . . .	5
3. Les interfaces disponibles . . . . .	5
3.1 Terminal CLI (interface principale) . . . . .	5
3.2 VS Code Extension . . . . .	5
3.3 JetBrains Plugin . . . . .	5
3.4 Desktop App . . . . .	6
3.5 Web (claude.ai/code) . . . . .	6
3.6 Slack . . . . .	6
3.7 Chrome Extension . . . . .	6
4. Installation . . . . .	6
4.1 Installation (methodes recommandees) . . . . .	6
4.2 Configuration de la cle API . . . . .	6
4.3 Verification de l'installation . . . . .	7
4.4 Installation Desktop . . . . .	7
4.5 Installation extensions IDE . . . . .	7
5. Interface et commandes essentielles . . . . .	7
5.1 Les commandes slash . . . . .	7
5.2 Raccourcis clavier . . . . .	8
5.3 La status line . . . . .	8
6. Extended Thinking . . . . .	8
6.1 Concept . . . . .	8
6.2 Fonctionnement automatique (Opus 4.6) . . . . .	9
6.3 Mots-cles heritage (encore fonctionnels) . . . . .	9
6.4 /effort — Controle programmatique . . . . .	9
6.5 Quand intervenir? . . . . .	10
7. Les modeles disponibles . . . . .	10
7.1 Vue d'ensemble . . . . .	10
7.2 Sonnet 4.6 (modele par default) . . . . .	10
7.3 Opus 4.6 (flagship) . . . . .	11
7.4 Haiku 4.5 (leger) . . . . .	11
7.5 Changer de modele . . . . .	11
8. Coûts et tokens . . . . .	11
8.1 Comprendre les tokens . . . . .	11
8.2 Monitoring des couts . . . . .	11

8.3 Optimiser ses couts . . . . .	12
8.4 Context window et compaction . . . . .	12
Exercice pratique : Premier pas avec Claude Code (30 min) . . . . .	12
Etape 1 : Installation (5 min) . . . . .	12
Etape 2 : Premiere interaction (5 min) . . . . .	13
Etape 3 : Premier prompt (5 min) . . . . .	13
Etape 4 : Extended Thinking (10 min) . . . . .	13
Etape 5 : Changer de modele (5 min) . . . . .	13
Points Cles a Retenir . . . . .	14

## Module 1 : Introduction a Claude Code

**Duree** : 1h30 **Prerequis** : Terminal de base, Node.js 18+ **Prochain module** : Module 2 - CLAUDE.md et Configuration

### Objectifs

A l'issue de ce module, vous serez capable de :

1. Comprendre ce qu'est Claude Code et son positionnement dans l'écosystème IA
2. Différencier Claude Code des autres outils IA pour le développement
3. Installer et configurer Claude Code sur votre poste
4. Utiliser les commandes essentielles de l'interface
5. Comprendre et utiliser l'Extended Thinking
6. Choisir le bon modèle selon votre besoin
7. Monitorer vos coûts et votre consommation de tokens

## 1. Qu'est-ce que Claude Code ?

### 1.1 Definition

Claude Code est le **CLI agentique officiel** d'Anthropic. Ce n'est pas un simple chatbot dans le terminal : c'est un agent autonome capable de lire, écrire, exécuter du code et naviguer dans votre codebase de manière indépendante.

Quelques chiffres clés : - **4% des commits GitHub** sont désormais générés via Claude Code (source : SemiAnalysis, février 2026) - Le SDK a été renommé "**Agent SDK**" pour refléter cette approche agentique - Claude Code peut gérer des tâches complexes impliquant des dizaines de fichiers en une seule session

### 1.2 Approche agentique

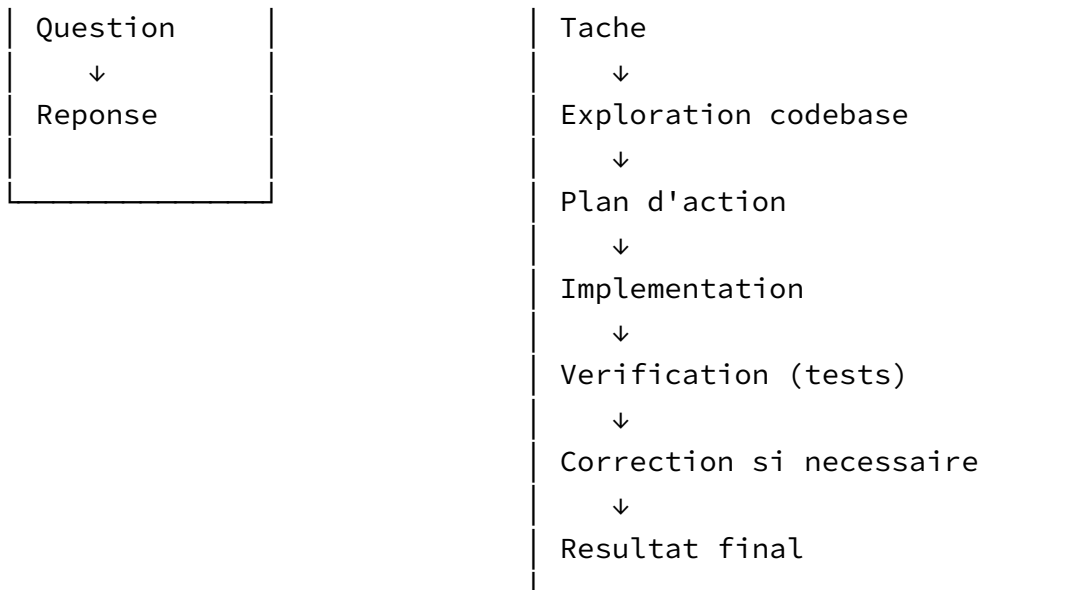
Contrairement à un assistant passif qui répond à des questions, Claude Code : - **Explore** votre codebase de manière autonome (lecture de fichiers, recherche de patterns) - **Planifie** ses actions avant de les exécuter - **Exécute** des commandes shell, crée et modifie des fichiers - **Vérifie** ses modifications en exécutant les tests - **Itère** jusqu'à ce que la tâche soit complétée

Assistant classique

\_\_\_\_\_

Agent Claude Code

\_\_\_\_\_



## 2. Comparaison avec les autres outils

### 2.1 Tableau comparatif

Critere	Claude Code	GitHub Copilot	ChatGPT	Cursor
<b>Type</b>	Agent CLI autonome	Autocompletion + Chat	Chatbot	IDE complet
<b>Execution de code</b>	Oui (natif)	Non	Non (sauf plugins)	Oui
<b>Lecture codebase</b>	Autonome, multi-fichiers	Fichier ouvert	Copier/coller	Projet ouvert
<b>Modification fichiers</b>	Directe, multi-fichiers	Suggestions inline	Non	Directe
<b>Commandes shell</b>	Oui	Non	Non	Oui
<b>Workflow Git</b>	Complet (branch, commit, PR)	Non	Non	Partiel
<b>Context window</b>	Jusqu'a 1M tokens (Opus)	Limite	128K (GPT-4)	Variable
<b>Mode headless</b>	Oui (scripting, CI/CD)	Non	API	Non

Critere	Claude Code	GitHub Copilot	ChatGPT	Cursor
<b>Prix</b>	Pay-per-use / Max plan	Abonnement	Abonnement	Abonnement
<b>Open source</b>	Oui (Apache 2.0)	Non	Non	Non

## 2.2 Quand utiliser quoi ?

- **Claude Code** : Taches complexes, refactoring multi-fichiers, exploration codebase, automatisation, CI/CD
- **Copilot** : Autocompletion rapide en codant, suggestions ligne par ligne
- **ChatGPT** : Questions generales, brainstorming, explication de concepts
- **Cursor** : Developpement interactif avec UI riche, edition visuelle

## 3. Les interfaces disponibles

Claude Code est disponible sur **7 interfaces** differentes :

### 3.1 Terminal CLI (interface principale)

L'interface native. C'est la plus puissante et la plus flexible.

```
# Lancer Claude Code
```

```
claude
```

```
# Lancer avec un prompt direct
```

```
claude "Explique-moi l'architecture de ce projet"
```

```
# Mode headless (non interactif)
```

```
claude -p "Liste les fichiers TypeScript du projet"
```

### 3.2 VS Code Extension

Extension officielle integree a Visual Studio Code. Permet d'utiliser Claude Code directement dans l'editeur avec un panneau lateral dedie.

### 3.3 JetBrains Plugin

Plugin pour tous les IDE JetBrains (IntelliJ, WebStorm, PhpStorm, PyCharm, etc.).

### 3.4 Desktop App

Application de bureau standalone disponible sur macOS, Windows et Linux.

### 3.5 Web (claude.ai/code)

Interface web accessible depuis n'importe quel navigateur, connectee a un repository GitHub.

### 3.6 Slack

Integration Slack pour utiliser Claude Code dans des channels d'equipe.

### 3.7 Chrome Extension

Extension Chrome pour l'automatisation web et les tests QA avec Claude Code.

---

## 4. Installation

### 4.1 Installation (methodes recommandees)

```
# macOS (Homebrew — recommande)
brew install claude-code

# Linux / macOS (curl)
curl -fsSL https://cli.anthropic.com/install.sh | sh

# Windows (WinGet)
winget install Anthropic.ClaudeCode

# npm (fallback — officiellement deprecie)
npm install -g @anthropic-ai/claude-code

# Verification de l'installation
claude --version
```

**Note :** Anthropic recommande les methodes d'installation natives (Homebrew, curl, WinGet).  
L'installation via npm reste fonctionnelle mais est officiellement depreciee.

### 4.2 Configuration de la cle API

```
# Premiere utilisation : Claude vous guide dans la configuration
claude
```

```
# Ou configurer manuellement la variable d'environnement
export ANTHROPIC_API_KEY="sk-ant-..."
```

**Note :** Avec le plan Max d'Anthropic (abonnement mensuel), la cle API n'est pas necessaire. L'authentification se fait via votre compte Anthropic.

### 4.3 Verification de l'installation

```
# Verifier la version
claude --version

# Lancer Claude Code et verifier le fonctionnement
claude "Dis-moi bonjour"
```

### 4.4 Installation Desktop

Telecharger l'application depuis [claude.ai/download](https://claude.ai/download) et suivre les instructions d'installation pour votre OS.

### 4.5 Installation extensions IDE

#### VS Code :

Extensions → Rechercher "Claude Code" → Installer

#### JetBrains :

Settings → Plugins → Marketplace → "Claude Code" → Installer

---

## 5. Interface et commandes essentielles

### 5.1 Les commandes slash

Une fois dans Claude Code, vous avez acces a des commandes prefixees par / :

Commande	Description	Usage
/help	Affiche l'aide et les commandes disponibles	Navigation



Commande	Description	Usage
<code>/status</code>	Affiche l'état actuel (modele, tokens, cout)	Monitoring
<code>/clear</code>	Vide le contexte de conversation	Gestion contexte
<code>/compact</code>	Compacte le contexte (resume)	Gestion contexte
<code>/exit</code> ou <code>/quit</code>	Quitte Claude Code	Navigation
<code>/cost</code>	Affiche le cout de la session	Monitoring
<code>/history</code>	Historique des sessions	Navigation
<code>/model</code>	Change de modele	Configuration
<code>/fast</code>	Bascule le mode rapide (Opus 4.6 2.5x, cout 6x)	Performance
<code>/plan</code>	Active le mode planification	Workflow
<code>/think</code>	Active la reflexion approfondie	Reflexion

## 5.2 Raccourcis clavier

Raccourci	Action
<code>Ctrl+C</code>	Annuler la generation en cours
<code>Esc</code>	Annuler / Revenir en arriere
<code>Esc+Esc</code>	Rewind au dernier checkpoint
<code>Shift+Tab</code>	Basculer en mode Plan
<code>Tab</code>	Accepter une suggestion

## 5.3 La status line

En bas de l'interface, la status line affiche en permanence : - Le **modele** actif - Le **pourcentage de contexte** utilise - Le **cout** de la session - Le **mode** actuel (normal, plan, fast)

## 6. Extended Thinking

### 6.1 Concept

L'Extended Thinking permet a Claude de "reflechir plus longtemps" avant de repondre. Plus le budget de reflexion est eleve, plus Claude analyse en profondeur le probleme.

## 6.2 Fonctionnement automatique (Opus 4.6)

Depuis Opus 4.6, l'Extended Thinking est **automatique**. Le modele ajuste lui-meme la profondeur de sa reflexion en fonction de la complexite de la tache. Il n'est plus necessaire de demander explicitement une reflexion approfondie.

## 6.3 Mots-cles heritage (encore fonctionnels)

Les mots-cles de reflexion (think, think hard, think harder, ultrathink) sont **deprecies** mais restent fonctionnels dans Claude Code CLI. Ils peuvent encore etre utilises pour forcer un niveau de reflexion :

Mot-cle	Effet	Statut
think	Reflexion legere	Deprecie
think hard	Reflexion moderee	Deprecie
think harder	Reflexion elevee	Deprecie
ultrathink	Reflexion maximale	Deprecie

**Recommandation :** Laissez le modele gerer automatiquement sa reflexion. Les mots-cles ne sont plus la methode recommandee.

## 6.4 /effort — Controle programmatique

La commande /effort (ou le flag --effort) permet de controler la profondeur de reflexion de maniere plus granulaire :

```
# En mode interactif
/effort high
```

```
# En mode headless
claude -p "Renomme cette variable" --effort low
claude -p "Analyse cette architecture" --effort high
```

Niveau	Usage
low	Taches simples, rapides, economiques
medium	Taches courantes (default)
high	Problemes complexes, architecture, debugging

## 6.5 Quand intervenir?

Tache simple (renommer variable, ajouter import)

→ Laisser le default (automatique) ou `--effort low`

Tache moderee (implementer une fonction, corriger un bug)

→ Laisser le default (automatique)

Tache complexe (refactoring, nouvelle architecture, bug subtil)

→ `--effort high` ou laisser Opus gerer automatiquement

En mode headless / scripting

→ Utiliser `--effort` pour controler les couts

## 7. Les modeles disponibles

### 7.1 Vue d'ensemble

Modele	ID	Caracteristiques	Prix (input/output)
<b>Sonnet 4.6</b>	<code>claude-sonnet-4-6</code>	Rapide, bon rapport qualite/prix	\$3 / \$15 par M tokens
<b>Opus 4.6</b>	<code>claude-opus-4-6</code>	Flagship, le plus intelligent	\$5 / \$25 par M tokens
<b>Haiku 4.5</b>	<code>claude-haiku-4-5-20251001</code>	Leger, tres economique	\$1 / \$5 par M tokens

### 7.2 Sonnet 4.6 (modele par default)

- **Vitesse** : Rapide
- **Intelligence** : Tres bonne pour la majorite des taches
- **Context window** : 200K tokens (1M en beta)
- **Cas d'usage** : Developpement quotidien, generation de code, refactoring

### 7.3 Opus 4.6 (flagship)

- **Vitesse** : Plus lent (sauf en mode /fast)
- **Intelligence** : Maximum, raisonnement avance
- **Context window** : 1M tokens
- **Output max** : 128K tokens
- **Adaptive thinking** : Ajuste automatiquement sa reflexion
- **Cas d'usage** : Architecture complexe, debugging difficile, decisions critiques

### 7.4 Haiku 4.5 (leger)

- **Vitesse** : Tres rapide
- **Intelligence** : Bonne pour les taches simples
- **Context window** : 200K tokens
- **Cas d'usage** : Taches simples, autocompletion, scripts rapides, economie de couts

### 7.5 Changer de modele

*# Via la commande /model*

`/model opus`

*# Via la commande /fast (toggle Opus rapide)*

`/fast`

*# Via la ligne de commande*

`claude --model opus "Ma demande"`

---

## 8. Couts et tokens

### 8.1 Comprendre les tokens

Un token represente environ 3/4 d'un mot en anglais (un peu moins en francais). Chaque interaction avec Claude consomme des tokens : - **Tokens d'entree (input)** : Votre prompt + contexte envoye - **Tokens de sortie (output)** : La reponse de Claude - **Context window** : Taille maximale de la conversation (200K a 1M tokens selon le modele)

### 8.2 Monitoring des couts

*# Afficher le cout de la session en cours*

`/cost`

*# Afficher le statut complet (modele + tokens + cout)*

```
/status
```

```
# En mode headless, afficher les couts
claude -p "Mon prompt" --output-format json
# Le JSON inclut les informations de cout
```

### 8.3 Optimiser ses couts

Strategie	Impact
Utiliser Sonnet 4.6 pour les taches courantes	Economie importante
Utiliser /clear entre les taches	Reduit le contexte envoye
Utiliser /compact quand le contexte grossit	Comprime le contexte
Etre precis dans ses prompts	Moins d'iterations
Mode headless pour les scripts	Pas de contexte accumule
Haiku 4.5 pour les taches simples	Cout minimal
Eviter le Fast Mode sauf besoin urgent	6x le prix normal

### 8.4 Context window et compaction

La context window est la quantite totale de tokens que Claude peut “voir” en une fois :

Context Window (200K-1M)	
CLAUDE.md + Rules	(~5K tokens)
Conversation precedente	(~50K tokens)
Fichiers lus	(~30K tokens)
Prompt actuel	(~1K tokens)
→ Espace restant	(~114K tokens)

Quand le contexte approche la limite, Claude Code **compacte automatiquement** : il resume les messages anciens pour liberer de l'espace.

## Exercice pratique : Premier pas avec Claude Code (30 min)

### Etape 1 : Installation (5 min)

```
# Installer Claude Code (methode recommandee)
```

```
# macOS: brew install claude-code
# Linux: curl -fsSL https://cli.anthropic.com/install.sh | sh
# Windows: winget install Anthropic.ClaudeCode
# npm (fallback): npm install -g @anthropic-ai/claude-code

# Verifier l'installation
claude --version
```

## Etape 2 : Premiere interaction (5 min)

```
# Lancer Claude Code
claude

# Tester les commandes de base
/help
/status
/cost
```

## Etape 3 : Premier prompt (5 min)

```
# Demander a Claude de se presenter
> Presente-toi et explique ce que tu peux faire

# Explorer les capacites
> Quels fichiers existent dans le repertoire courant ?

# Tester l'execution de commandes
> Quelle version de Node.js est installee sur ce systeme ?
```

## Etape 4 : Extended Thinking (10 min)

```
# Prompt simple (sans thinking)
> Quels sont les avantages de TypeScript ?

# Prompt avec thinking
> think about the pros and cons of microservices vs monolith for a startup

# Prompt avec thinking approfondi
> think hard about how to design a scalable event-driven architecture
```

Observez la difference de profondeur et de qualite entre les reponses.

## Etape 5 : Changer de modele (5 min)

```
# Verifier le modele actuel
/status
```

*# Passer en Opus*

/model opus

*# Poser une question complexe*

> Analyse les trade-offs entre REST, GraphQL et gRPC pour une API publique

*# Revenir a Sonnet*

/model sonnet

*# Tester le mode fast*

/fast

---

## Points Cles a Retenir

1. **Claude Code est un agent**, pas un simple chatbot : il explore, planifie, execute et verifie
2. **7 interfaces** disponibles : CLI, VS Code, JetBrains, Desktop, Web, Slack, Chrome
3. **Extended Thinking** est automatique avec Opus 4.6. Utilisez /effort (low/medium/high) pour un controle explicite
4. **3 modeles** : Sonnet 4.6 (quotidien), Opus 4.6 (complexe), Haiku 4.5 (economique)
5. **Monitorer ses couts** avec /cost et /status
6. **La context window est une ressource finie** : utiliser /clear et /compact pour la gerer
7. **Le mode headless** (claude -p) permet l'integration dans des scripts et pipelines

---

**Prochain module** : Module 2 - CLAUDE.md et Configuration