Plan d'apprentissage complet Rust

De débutant à expert en 14 semaines

Semaine 1 : Fondamentaux de Rust

Jour 1 : Installation et Hello World

Objectif : Configurer l'environnement Rust

- Installer Rust via rustup
- Comprendre cargo (gestionnaire de paquets)
- Créer un projet avec `cargo new`
- Structure d'un projet Rust
- Compiler et exécuter avec `cargo run`
- `cargo build`, `cargo check`, `cargo fmt`

Exercice : Créer un projet "hello_rust" qui affiche un message de bienvenue personnalisé avec plusieurs println! utilisant différents types de formatage.

Jour 2 : Variables et mutabilité

Objectif : Comprendre l'ownership de base

- Variables immutables par défaut
- Mot-clé `mut` pour la mutabilité
- Shadowing de variables
- Différence entre `const` et variables immutables
- Convention de nommage (snake_case)

Exercice : Créer un programme qui démontre shadowing : déclarer une variable, la "shadower" avec un nouveau type, et montrer la différence avec mut.

Jour 3 : Types de données scalaires

Objectif : Maîtriser les types primitifs

- Entiers : i8, i16, i32, i64, i128, u8, u16, u32, u64, u128
- Flottants : f32, f64
- Booléens : bool
- Caractères : char (Unicode)
- Inférence de types
- Annotations de types explicites

Exercice : Créer un programme qui manipule tous les types scalaires, effectue des conversions entre types, et affiche les limites de chaque type numérique.

Jour 4 : Types composés

Objectif : Tuples et arrays

- Tuples : création, accès, déstructuration
- Arrays : taille fixe, type homogène
- Accès aux éléments
- Slices (introduction)
- Pattern matching avec tuples

Exercice : Créer une fonction qui retourne un tuple (min, max, moyenne) d'un array de nombres. Déstructurer le résultat et l'afficher.

Jour 5 : Fonctions

Objectif : Définir et utiliser des fonctions

- Déclaration avec `fn`
- Paramètres et types
- Valeurs de retour (avec et sans `return`)

- Expressions vs statements
- Fonctions dans d'autres fonctions
- Diverging functions (!)

Exercice : Créer un ensemble de fonctions mathématiques (puissance, factorielle, fibonacci) et une fonction qui les teste toutes.

Jour 6 : Structures de contrôle

Objectif : if, loop, while, for

- Expressions if/else (retournent une valeur)
- loop avec break et continue
- while loops
- for loops avec ranges
- Labels pour boucles imbriquées

Exercice : Créer un jeu de devinette de nombre : le programme génère un nombre aléatoire, l'utilisateur devine, reçoit des indices (trop haut/bas) jusqu'à trouver.

Jour 7 : Projet Semaine 1 - Convertisseur de températures

Projet complet : Application CLI interactive

- Menu de choix : Celsius ↔ Fahrenheit, Celsius ↔ Kelvin
- Fonctions de conversion pour chaque direction
- Validation des entrées utilisateur
- Boucle permettant plusieurs conversions
- Option pour quitter
- Gestion basique des erreurs (parse)
- Formatage élégant des résultats

Semaine 2 : Ownership et gestion de la mémoire

Jour 8 : Ownership - Les règles

Objectif : Comprendre le concept central de Rust

- Les trois règles d'ownership
- Scope et durée de vie
- Move semantics
- Copy trait vs Move
- Drop trait

Exercice : Créer des exemples qui démontrent les moves, et observer quand une variable n'est plus utilisable. Expérimenter avec des types qui implement Copy.

Jour 9 : References et Borrowing

Objectif : Emprunter sans prendre ownership

- References immutables (&T)
- References mutables (&mut T)
- Règles du borrowing
- Dangling references (prévention)
- Multiples emprunts immutables vs un seul mutable

Exercice : Créer des fonctions qui calculent la longueur d'un String sans en prendre ownership, modifient un String en place avec &mut, et démontrent les règles de borrowing.

Jour 10 : Slices

Objectif : Références vers des portions de données

- String slices (&str)
- Array slices

- Slices mutables
- Relation entre String et &str
- String literals comme slices

Exercice : Créer une fonction qui prend un String et retourne le premier mot (slice). Créer une fonction qui inverse l'ordre des mots dans une phrase.

Jour 11 : Structs

Objectif : Types de données personnalisés

- Définition de structs
- Syntaxe d'initialisation
- Field init shorthand
- Struct update syntax
- Tuple structs
- Unit-like structs

Exercice : Créer une struct `Rectangle` avec largeur et hauteur. Créer plusieurs instances avec différentes syntaxes. Créer une struct `Color` en tuple struct.

Jour 12 : Méthodes et fonctions associées

Objectif : Comportement associé aux structs

- Blocs `impl`
- Méthodes avec `&self`, `&mut self`, `self`
- Associated functions (sans self)
- Multiple impl blocks
- Méthodes avec paramètres additionnels

Exercice : Ajouter des méthodes à Rectangle : area(), can_hold(autre), set_width(), new() (associated function). Créer une struct Circle avec les mêmes concepts.

Jour 13 : Enums et Pattern Matching

Objectif : Types avec variantes

- Définition d'enums
- Variants avec données
- Option<T> enum
- match expressions
- Exhaustivité du matching
- Placeholder _ et variable binding

Exercice : Créer un enum `Message` avec variantes (Quit, Move{x,y}, Write(String), ChangeColor(i32,i32,i32)). Implémenter une méthode call() qui match sur chaque variante.

Jour 14 : Projet Semaine 2 - Système de gestion de bibliothèque

Projet complet : Application de bibliothèque personnelle

- Struct Book (titre, auteur, ISBN, statut)
- Enum BookStatus (Available, Borrowed(String), Reserved)
- Struct Library avec Vec de Books
- Méthodes : add_book, find_by_title, borrow_book, return_book
- Pattern matching pour gérer les statuts
- Associated function pour créer une Library
- Menu interactif CLI
- Affichage formaté de tous les livres

Semaine 3 : Collections et gestion des erreurs

Jour 15 : Vectors

Objectif : Collections dynamiques

- Création avec Vec::new() et vec! macro
- Push et pop
- Indexation vs get()
- Itération (immutable et mutable)
- Capacity vs length
- Vecteurs de different types (avec enums)

Exercice : Créer un programme qui stocke des scores, calcule moyenne/médiane/mode. Gérer l'accès sécurisé avec get() et gérer les Options retournées.

Jour 16 : Strings

Objectif : Maîtriser les strings en Rust

- String vs &str
- Création et mise à jour
- Concatenation (+ et format!)
- Indexation (pourquoi c'est complexe)
- Itération : chars() vs bytes()
- Slicing sécurisé

Exercice : Créer un analyseur de texte qui compte voyelles/consonnes, mots, lignes. Implémenter une fonction de "pig latin" transformation.

Jour 17 : HashMaps

Objectif : Dictionnaires en Rust

- Création de HashMap
- Insert et get
- Entry API pour update

- Ownership avec HashMaps
- Itération sur clés/valeurs
- or_insert et and_modify

Exercice : Créer un compteur de mots : lire du texte et compter la fréquence de chaque mot dans une HashMap. Afficher les 5 mots les plus fréquents.

Jour 18 : Result<T, E>

Objectif : Gestion d'erreurs récupérables

- Le type Result
- Propagation d'erreurs avec ?
- unwrap et expect (et pourquoi les éviter)
- match sur Result
- Créer ses propres types d'erreur

Exercice : Créer une fonction qui lit un fichier et parse son contenu en nombres. Gérer toutes les erreurs possibles (fichier inexistant, contenu invalide) avec Result.

Jour 19 : panic! et erreurs irrécupérables

Objectif : Comprendre quand paniquer

- Macro panic!
- Backtraces
- Quand utiliser panic vs Result
- unwrap et expect appropriés
- Tests avec should_panic

Exercice : Créer un programme qui valide des données critiques et panic! avec messages descriptifs si les invariants sont violés. Écrire des tests pour ces panics.

Jour 20 : Iterators

Objectif : Programmation fonctionnelle

- Le trait Iterator
- Méthodes : map, filter, fold, collect
- Iterators sont lazy
- Chaînage d'adaptateurs
- Créer ses propres iterators

Exercice : Résoudre des problèmes avec iterators : filtrer nombres pairs, transformer en carrés, sommer résultats. Comparer performance vs loops.

Jour 21 : Projet Semaine 3 - Analyseur de logs

Projet complet : Outil d'analyse de fichiers de logs

- Lire un fichier de logs (format : timestamp, level, message)
- Parser chaque ligne en struct LogEntry
- Stocker dans Vec
- Statistiques : compter par level (ERROR, WARN, INFO)
- HashMap pour messages d'erreur uniques
- Filter par date range
- Chercher par mot-clé
- Gestion complète des erreurs avec Result
- Tests pour les fonctions de parsing
- Output formaté ou JSON

Semaine 4 : Traits et génériques

Jour 22 : Generics

Objectif : Code réutilisable avec types génériques

- Generic functions
- Generic structs
- Generic enums
- Multiple type parameters
- Contraintes avec trait bounds
- Monomorphization

Exercice : Créer une struct Point<T> générique. Implémenter des méthodes qui fonctionnent avec n'importe quel type. Créer une fonction largest<T> qui trouve le plus grand élément.

Jour 23 : Traits - Définition

Objectif : Définir des comportements partagés

- Définir un trait
- Implémenter un trait
- Default implementations
- Traits comme paramètres
- Trait bounds
- Traits et lifetimes

Exercice : Créer un trait Summary avec méthode summarize(). L'implémenter pour NewsArticle et Tweet. Créer une fonction qui accepte n'importe quel type implementant Summary.

Jour 24 : Traits standards

Objectif : Traits importants de la std lib

- Display et Debug
- Clone et Copy
- Default
- PartialEq et Eq
- PartialOrd et Ord
- Derive macros

Exercice : Créer des structs et dériver les traits appropriés. Implémenter Display manuellement. Créer une struct orderable et l'utiliser dans un BTreeSet.

Jour 25 : Traits avancés

Objectif : Concepts avancés

- Associated types
- Default type parameters
- Operator overloading
- Calling methods même nom
- Supertraits
- Newtype pattern

Exercice : Implémenter Add trait pour additionner deux structs personnalisés. Créer un trait avec associated type et l'implémenter pour plusieurs types.

Jour 26 : Lifetimes - Introduction

Objectif : Annotations de durée de vie

- Pourquoi les lifetimes existent
- Syntaxe des lifetimes

- Lifetime annotations dans fonctions
- Lifetime elision rules
- Lifetimes dans structs

Exercice : Créer des fonctions qui retournent la plus longue de deux string slices. Créer une struct qui contient une référence et annoter correctement le lifetime.

Jour 27 : Lifetimes - Avancé

Objectif : Cas complexes

- Multiple lifetime parameters
- Lifetime bounds
- 'static lifetime
- Lifetime subtyping
- Generic type parameters, trait bounds, et lifetimes ensemble

Exercice : Créer une struct complexe avec plusieurs références de lifetimes différents. Implémenter des méthodes avec annotations de lifetime appropriées.

Jour 28 : Projet Semaine 4 - Mini système de base de données en mémoire

Projet complet : Base de données générique

- Trait Database<T> générique
- Implémentations pour InMemoryDB
- Opérations : insert, get, update, delete, query
- Trait Query pour différents types de requêtes
- Generic constraints avec trait bounds
- Lifetime annotations pour references
- Index secondaire avec HashMap
- API fluent avec builder pattern

- Tests exhaustifs
- Documentation avec exemples

Semaine 5 : Tests et documentation

Jour 29 : Tests unitaires

Objectif : Tester son code

- Module tests avec #[cfg(test)]
- Macro assert!, assert_eq!, assert_ne!
- #[test] attribute
- Tests qui paniquent avec #[should_panic]
- Result<T, E> dans tests
- cargo test et ses options

Exercice : Reprendre les fonctions mathématiques précédentes et écrire une suite de tests complète. Tests pour cas normaux et edge cases.

Jour 30 : Tests d'intégration

Objectif : Tester l'API publique

- Directory tests/
- Tests d'intégration vs unitaires
- Common modules dans tests
- Tests de binary crates
- Organisation des tests

Exercice : Créer une petite bibliothèque et écrire des tests d'intégration qui testent son API publique comme l'utiliserait un client.

Jour 31 : Documentation

Objectif : Documenter efficacement

- Doc comments avec ///
- //! pour module/crate docs
- Markdown dans documentation
- Sections (Examples, Panics, Errors, Safety)
- cargo doc
- Tests dans documentation

Exercice : Documenter complètement une bibliothèque avec exemples exécutables dans la doc. Vérifier que les exemples passent avec cargo test.

Jour 32 : Benchmarking

Objectif : Mesurer les performances

- Criterion.rs pour benchmarks
- Configurer benchmarks
- Comparer implémentations
- Interpréter les résultats
- Profiling basique

Exercice : Benchmarker différentes approches pour un même problème (ex: recherche dans Vec vs HashSet, différentes méthodes de string concatenation).

Jour 33 : Property-based testing

Objectif : Tests avec génération automatique

- Concept de property testing
- Crate proptest ou quickcheck
- Définir des propriétés
- Generators et strategies

- Shrinking

Exercice : Écrire des property tests pour une fonction de tri personnalisée. Définir des propriétés (idempotence, ordre, longueur préservée).

Jour 34 : Fuzzing

Objectif : Découvrir des bugs cachés

- Cargo-fuzz
- Définir des fuzz targets
- Interpréter les crashs
- Corpus et minimisation
- Fuzzing continu

Exercice : Créer un parser simple (JSON ou autre format) et le fuzzer pour découvrir des inputs qui causent des panics ou erreurs.

Jour 35 : Projet Semaine 5 - Bibliothèque de parsing avec TDD

Projet complet : Parser de format personnalisé (CSV ou INI)

- Développement TDD : tests d'abord
- Parser complet avec gestion d'erreurs
- Tests unitaires pour chaque fonction
- Tests d'intégration pour parsing complet
- Property tests pour invariants
- Documentation exhaustive avec exemples
- Benchmarks pour mesurer performances
- Code coverage >90%
- README avec guide d'utilisation

Semaine 6 : Modules et organisation

Jour 36 : Modules et crates

Objectif : Organiser le code

- Système de modules
- mod keyword
- Fichiers et dossiers
- Chemins absolus vs relatifs
- pub keyword et visibilité
- use keyword

Exercice : Créer un projet avec plusieurs modules organisés hiérarchiquement. Expérimenter avec différentes structures (inline, fichiers séparés, dossiers).

Jour 37 : Packages et workspaces

Objectif : Projets multi-crates

- Cargo.toml et manifests
- Binary vs library crates
- Workspaces avec plusieurs crates
- Dépendances locales
- Path dependencies

Exercice : Créer un workspace avec 3 crates : une bibliothèque commune, et deux binaires qui l'utilisent. Organiser les dépendances correctement.

Jour 38 : Publication sur crates.io

Objectif : Partager son code

- Préparer une crate pour publication
- Metadata dans Cargo.toml

- Documentation et README
- Versioning sémantique
- cargo publish
- Badges et CI

Exercice : Préparer une petite bibliothèque utilitaire pour publication (sans publier réellement). S'assurer que la doc est complète et que les tests passent.

Jour 39 : Build scripts

Objectif : Customiser le build

- build.rs files
- Générer du code au build
- Compiler des dépendances C
- Environment variables
- Cargo features

Exercice : Créer un build script qui génère du code Rust à partir d'un fichier de configuration. Utiliser cargo features pour activer/désactiver du code.

Jour 40 : Macros déclaratives

Objectif : macro_rules!

- Syntaxe des macros
- Pattern matching dans macros
- Repetitions
- Hygiène des macros
- Debugging de macros

Exercice : Créer une macro hashmap! similaire à vec! qui crée une HashMap. Créer une macro assert_in_range! pour les tests.

Jour 41 : Macros procédurales - Introduction

Objectif : Macros plus puissantes

- Types de proc macros
- Derive macros
- Attribute macros
- Function-like macros
- syn et quote crates

Exercice : Créer une derive macro simple qui implémente un trait personnalisé. Tester avec plusieurs structs.

Jour 42 : Projet Semaine 6 - Framework de serialization personnalisé

Projet complet : Mini framework avec macros

- Trait Serialize
- Derive macro #[derive(Serialize)]
- Serialization en format personnalisé (key=value)
- Support pour structs, enums, nested types
- Workspace : bibliothèque + derive macro crate
- Tests exhaustifs pour la macro
- Documentation avec exemples
- Benchmarks vs serde

Semaine 7 : Closures et itérateurs avancés

Jour 43 : Closures - Basics

Objectif : Fonctions anonymes

- Syntaxe des closures
- Inférence de types
- Capture de l'environnement
- Fn, FnMut, FnOnce traits
- Move closures

Exercice : Créer des exemples de closures avec différents types de capture. Implémenter une fonction de cache qui utilise une closure pour calculer valeurs.

Jour 44 : Closures avancées

Objectif : Cas d'usage complexes

- Returning closures
- Closures dans structs
- Lifetime considerations
- Box<dyn Fn> vs impl Fn
- Closures et ownership

Exercice : Créer une struct qui stocke plusieurs closures et les exécute en séquence. Implémenter un système de callbacks.

Jour 45 : Iterator trait

Objectif : Créer ses propres iterators

- Implémenter Iterator trait
- IntoIterator trait
- Iterator adapters personnalisés
- Infinite iterators
- Peekable iterators

Exercice : Créer un iterator Fibonacci. Créer un iterator qui parcourt un arbre binaire. Implémenter IntoIterator pour une struct personnalisée.

Jour 46 : Iterator patterns

Objectif : Patterns avancés

- Consumers: collect, fold, any, all
- Adapters: map, filter, filter_map, flat_map
- Chaînage complexe
- Performance avec iterators
- Iterator vs loops (quand utiliser quoi)

Exercice : Résoudre des problèmes algorithmiques uniquement avec iterators : filtrer/transformer/agréger des données complexes.

Jour 47 : Functional programming

Objectif : Paradigme fonctionnel en Rust

- Higher-order functions
- Function composition
- Immutabilité
- Pattern matching avancé
- Option et Result comme monads

Exercice : Refactoriser du code impératif en style fonctionnel. Composer plusieurs fonctions pour créer un pipeline de traitement de données.

Jour 48 : Lazy evaluation

Objectif : Évaluation paresseuse

- Lazy_static crate
- OnceCell et OnceLock

- Computation on-demand
- Memoization patterns
- Performance considerations

Exercice : Implémenter un système de cache lazy : calcul à la première demande, puis réutilisation. Benchmarker l'amélioration de performance.

Jour 49 : Projet Semaine 7 - Bibliothèque de traitement de données

Projet complet : Pipeline de data processing

- API fluent avec iterators
- Opérations : filter, map, reduce, group_by, sort
- Custom iterators pour chaque opération
- Lazy evaluation
- Support pour parallel processing (introduction)
- Closures pour transformations customisées
- Tests avec property testing
- Benchmarks de performance
- Documentation avec exemples complexes

Semaine 8 : Smart pointers et types avancés

Jour 50 : Box<T>

Objectif : Pointeur heap simple

- Quand utiliser Box
- Recursive types
- Trait objects avec Box

- Deref trait
- Drop trait

Exercice : Créer une liste chaînée (cons list) avec Box. Implémenter des méthodes push, pop, length.

Jour 51 : Rc<T> et Arc<T>

Objectif : Ownership partagé

- Reference counting
- Rc pour single-thread
- Arc pour multi-thread
- Weak references
- Cycles et memory leaks

Exercice : Créer un graphe avec Rc/Weak pour éviter les cycles. Démontrer le partage d'ownership entre plusieurs structs.

Jour 52 : RefCell<T> et interior mutability

Objectif : Mutabilité intérieure

- Pattern interior mutability
- RefCell et borrow checking runtime
- Cell<T> pour Copy types
- Rc<RefCell<T>> pattern
- Quand l'utiliser

Exercice : Créer un mock object pour tests qui track les appels avec RefCell. Implémenter un système de cache mutable partagé.

Jour 53 : Custom smart pointers

Objectif : Créer ses propres smart pointers

- Deref et DerefMut traits
- Drop trait customization
- Pointer metadata
- Smart pointer patterns

Exercice : Créer un smart pointer qui log toutes les opérations. Implémenter un counted pointer personnalisé avec statistiques.

Jour 54 : Unsafe Rust

Objectif : Comprendre unsafe

- Les 5 superpowers d'unsafe
- Raw pointers
- Unsafe functions
- Unsafe traits
- Extern functions (FFI introduction)
- Safe abstractions sur unsafe

Exercice : Implémenter une fonction unsafe qui manipule des raw pointers. Créer une safe abstraction autour. Comprendre quand c'est justifié.

Jour 55 : Types avancés

Objectif : Features de type system

- Newtype pattern
- Type aliases
- Never type (!)
- Dynamically sized types
- Function pointers
- Returning closures

Exercice : Utiliser newtype pattern pour type safety (UserId, Email, etc.). Créer des types aliases pour simplifier signatures complexes.

Jour 56 : Projet Semaine 8 - Allocateur mémoire personnalisé

Projet complet : Custom allocator

- Implémenter GlobalAlloc trait
- Pool allocator pour objets de taille fixe
- Statistics de allocations
- Unsafe code nécessaire
- Tests de correctness
- Benchmarks vs default allocator
- Documentation des garanties de sécurité
- Exemples d'utilisation

Semaine 9 : Concurrence

Jour 57 : Threads

Objectif : Concurrence avec threads

- thread::spawn
- JoinHandles
- Move closures dans threads
- thread::sleep
- Thread panic handling
- Channels pour communication

Exercice : Créer un programme qui spawne plusieurs threads pour calculer différentes parties d'un problème, puis combine les résultats.

Jour 58 : Message passing

Objectif : Channels pour communication

- mpsc channels
- Sending et receiving
- Multiple producers
- Channel et ownership
- Synchronous vs asynchronous channels

Exercice : Implémenter un producer-consumer pattern : plusieurs producers envoient des jobs, un consumer les traite et affiche les résultats.

Jour 59 : Shared state

Objectif : Partage avec mutex et atomics

- Mutex<T>
- Lock poisoning
- RwLock
- Atomic types
- Ordering (Relaxed, Acquire, Release, SeqCst)

Exercice : Créer un compteur partagé entre threads avec Mutex. Comparer performance avec AtomicUsize. Implémenter un cache concurrent avec RwLock.

Jour 60 : Sync et Send traits

Objectif : Thread safety garanties

- Marker traits Sync et Send
- Quels types sont Sync/Send
- Implementing Sync et Send (rare)
- Compiler enforcement

- Arc vs Rc

Exercice : Analyser quels types sont Sync/Send et pourquoi. Créer une struct et déterminer son thread-safety.

Jour 61 : Scoped threads

Objectif : Threads avec borrows

- thread::scope
- Borrowing dans scoped threads
- Lifetime advantages
- Use cases vs spawned threads

Exercice : Refactorer un programme qui clone des données pour les threads, en utilisant scoped threads pour emprunter à la place.

Jour 62 : Rayon pour data parallelism

Objectif : Parallélisme facile

- Rayon crate
- Par iterators
- par_iter() vs iter()
- Thread pools
- Configuration

Exercice : Paralléliser des opérations sur grandes collections avec Rayon. Comparer performance avec itérations séquentielles. Benchmarker différentes tailles de données.

Jour 63 : Projet Semaine 9 - Web Scraper concurrent

Projet complet : Scraper parallèle

- Crawler de sites web
- Thread pool pour requêtes HTTP

- Channels pour collecter résultats
- Shared state pour URLs visitées (Mutex<HashSet>)
- Rate limiting avec atomics
- Graceful shutdown
- Progress reporting en temps réel
- Statistiques finales
- Tests de concurrence
- Error handling dans contexte concurrent

Semaine 10 : Async/Await

Jour 64 : Async basics

Objectif : Introduction à async

- Futures trait
- async fn et async blocks
- await keyword
- Executors (tokio, async-std)
- Runtime vs threads

Exercice : Créer des fonctions async simples. Comprendre la différence entre déclarer async et exécuter avec runtime.

Jour 65 : Tokio runtime

Objectif : Runtime async populaire

- #[tokio::main]
- task::spawn

- Async tasks vs threads
- Blocking dans async
- tokio::time

Exercice : Créer plusieurs async tasks qui s'exécutent concurremment. Simuler des opérations I/O avec sleep. Mesurer le parallélisme.

Jour 66 : Async I/O

Objectif : File et network I/O async

- tokio::fs
- tokio::net
- Async read et write traits
- Buffered I/O
- Timeouts

Exercice : Créer un programme qui lit plusieurs fichiers en parallèle avec async I/O. Comparer temps d'exécution vs synchronous.

Jour 67 : Async HTTP avec reqwest

Objectif : HTTP client async

- reqwest crate
- Async requests
- Concurrent requests
- Connection pooling
- Error handling

Exercice : Fetcher plusieurs URLs en parallèle. Implémenter retry logic. Limiter concurrence avec semaphore.

Jour 68 : Streams

Objectif : Async iterators

- Stream trait
- stream! macro
- Adapter des futures en streams
- Combinateurs (map, filter, etc.)
- Backpressure

Exercice : Créer un stream qui génère des valeurs. Consumer le stream avec des transformations. Implémenter un stream personnalisé.

Jour 69 : Channels async

Objectif : Communication entre tasks

- tokio::sync::mpsc
- Bounded vs unbounded
- oneshot channels
- broadcast channels
- watch channels

Exercice : Implémenter un producer-consumer async. Multiple producers envoyant à multiple consumers via channels.

Jour 70 : Projet Semaine 10 - API REST async avec Axum

Projet complet : Web server complet

- Framework Axum
- Routes CRUD pour ressource
- Handlers async
- Shared state (Arc<Mutex<>>)

- Middleware (logging, auth)
- Database async (sqlx)
- Error handling
- Tests async
- Graceful shutdown
- Rate limiting
- Documentation OpenAPI

Semaine 11 : Web et networking

Jour 71 : HTTP server - Actix-web

Objectif : Alternative web framework

- Actix-web basics
- App state et data extractors
- Path, Query, Json extractors
- Middleware actix
- Responders

Exercice : Créer une API simple avec Actix-web. Comparer avec Axum : ergonomie, performance, style de code.

Jour 72 : WebSockets

Objectif : Communication bidirectionnelle

- tokio-tungstenite ou axum websockets
- Upgrade HTTP vers WS
- Send et receive messages

- Broadcast à multiples clients
- Heartbeat et reconnection

Exercice : Créer un serveur de chat WebSocket. Clients peuvent joindre des rooms, envoyer des messages qui sont broadcastés.

Jour 73 : gRPC avec tonic

Objectif : RPC moderne

- Protocol Buffers
- Définir services .proto
- tonic pour server et client
- Streaming (unary, server, client, bidirectional)
- Metadata et interceptors

Exercice : Créer un service gRPC simple : service Calculator. Implémenter client et server. Tester tous types de streaming.

Jour 74 : GraphQL avec async-graphql

Objectif : API GraphQL

- Schema definition
- Queries et mutations
- Resolvers
- Subscriptions
- DataLoader pattern

Exercice : Créer une API GraphQL pour un blog : posts, authors, comments. Implémenter queries, mutations, et relations entre types.

Jour 75 : Database avec sqlx

Objectif : Async SQL

- sqlx features
- Compile-time query checking
- Transactions
- Migrations
- Connection pooling

Exercice : Créer une application CRUD complète avec PostgreSQL/SQLite. Utiliser compile-time verification. Implémenter transactions pour opérations complexes.

Jour 76 : ORM avec Diesel

Objectif : Type-safe SQL

- Diesel setup
- Schema et migrations
- Query builder
- Associations
- Transactions

Exercice : Recréer l'application précédente avec Diesel. Comparer les approches : type safety, ergonomie, performance.

Jour 77 : Projet Semaine 11 - Plateforme de blogging complète

Projet complet : Application full-featured

- API REST avec Axum
- WebSocket pour notifications temps réel
- PostgreSQL avec sqlx
- Authentification JWT
- CRUD : users, posts, comments, likes
- Relations complexes

- Pagination et filtering
- Upload d'images
- Full-text search
- Rate limiting par user
- Middleware d'audit
- Tests d'intégration async
- Documentation API complète
- Docker compose avec services

Semaine 12 : Sécurité et cryptographie

Jour 78 : Cryptographie basics

Objectif : Primitives crypto

- ring ou rust-crypto
- Hashing (SHA-256, Blake3)
- Random number generation
- Constant-time comparisons
- Key derivation (PBKDF2, Argon2)

Exercice : Implémenter un système de stockage de passwords sécurisé avec salting et Argon2. Fonction de vérification avec timing attack protection.

Jour 79 : Symmetric encryption

Objectif : Chiffrement symétrique

- AES-GCM avec aead crate
- Nonces et IVs

- Authenticated encryption
- Encrypt et decrypt data
- Key management

Exercice : Créer un système de chiffrement de fichiers. Encrypt/decrypt avec mot de passe dérivé en clé. Gérer l'authentification.

Jour 80 : Asymmetric encryption

Objectif : Cryptographie à clé publique

- RSA basics
- Génération de paires de clés
- Signature et vérification
- Encryption/decryption
- x25519 et Ed25519

Exercice : Implémenter un système de signature de messages. Générer clés, signer un message, vérifier la signature.

Jour 81 : TLS et certificats

Objectif : Sécuriser les communications

- rustls pour TLS
- Certificats et CA
- HTTPS server
- Client certificate authentication
- mTLS

Exercice : Configurer un serveur HTTPS avec rustls. Générer certificats self-signed. Créer un client qui valide les certificats.

Jour 82 : JWT et OAuth2

Objectif : Authentification moderne

- jsonwebtoken crate
- Créer et valider JWTs
- Claims et payload
- oauth2 crate
- Flow OAuth2 complet

Exercice : Implémenter authentification JWT complète : login, token generation, validation, refresh tokens. Ajouter OAuth2 avec provider externe.

Jour 83 : Security best practices

Objectif : Sécurité applicative

- Input validation et sanitization
- SQL injection prevention
- XSS et CSRF
- Rate limiting avancé
- Audit logging
- Secrets management

Exercice : Audit d'une application existante. Identifier vulnérabilités potentielles. Implémenter les protections nécessaires.

Jour 84 : Projet Semaine 12 - Système de vault sécurisé

Projet complet : Gestionnaire de secrets

- Stockage chiffré de secrets (passwords, keys)
- Master password avec KDF
- Chiffrement AES-GCM par secret

- API avec authentification JWT
- Audit logs de tous les accès
- Rate limiting agressif
- TLS obligatoire
- Zero-knowledge proof concept
- Auto-lock après inactivité
- Tests de sécurité
- Documentation des menaces

Semaine 13 : Performance et optimisation

Jour 85 : Profiling CPU

Objectif : Identifier bottlenecks

- cargo-flamegraph
- perf sur Linux
- Instruments sur macOS
- Analyser les résultats
- Hot paths

Exercice : Profiler une application intensive en calcul. Identifier les fonctions les plus coûteuses. Optimiser et re-mesurer.

Jour 86 : Memory profiling

Objectif : Optimiser allocations

- valgrind et massif
- heaptrack

- cargo-bloat
- Reducing allocations
- Stack vs heap

Exercice : Profiler l'utilisation mémoire. Identifier allocations excessives. Utiliser techniques pour réduire (arena allocation, object pooling).

Jour 87 : SIMD et vectorisation

Objectif : Instructions parallèles

- std::simd (nightly)
- packed_simd
- Auto-vectorisation
- Manual SIMD
- Quand l'utiliser

Exercice : Implémenter des opérations vectorielles (somme, produit scalaire) avec et sans SIMD. Benchmarker les différences.

Jour 88 : Inline et compilation

Objectif : Optimisations compile-time

- #[inline] annotations
- Link-time optimization (LTO)
- Codegen units
- Profile-guided optimization
- Target CPU features

Exercice : Expérimenter avec différentes options de compilation. Mesurer impact sur taille binaire et performance.

Jour 89 : Concurrent data structures

Objectif : Structures lock-free

- crossbeam crate
- Lock-free queues
- Concurrent maps
- Epoch-based reclamation
- Compare-and-swap patterns

Exercice : Implémenter une queue lock-free simple. Benchmarker vs queue avec mutex. Comprendre les trade-offs.

Jour 90 : Benchmarking avancé

Objectif : Mesures précises

- Criterion.rs features avancées
- Statistical analysis
- Regression detection
- Flame graphs dans benchmarks
- CI pour benchmarks

Exercice : Créer une suite de benchmarks complète pour une bibliothèque. Configurer CI pour détecter les régressions de performance.

Jour 91 : Projet Semaine 13 - Moteur de traitement de données haute performance

Projet complet : Data processing engine

- Parser de CSV/JSON haute performance
- Pipeline de transformations
- Parallel processing avec Rayon
- SIMD pour opérations numériques

- Memory pooling pour allocations
- Streaming pour fichiers volumineux
- Benchmarks exhaustifs
- Profiling et optimisation documentée
- Comparaison avec outils existants
- Zero-copy optimizations

Semaine 14 : Projet final et sujets avancés

Jour 92 : FFI - Foreign Function Interface

Objectif : Interopérabilité avec C

- extern keyword
- C ABI
- Calling C depuis Rust
- Exposer Rust à C
- bindgen pour génération
- cbindgen pour headers

Exercice : Créer une bibliothèque Rust qui expose une API C. L'utiliser depuis un programme C. Wrapper une bibliothèque C depuis Rust.

Jour 93 : WebAssembly

Objectif : Rust pour le web

- wasm-pack
- Compiler pour wasm32
- wasm-bindgen

- Interfacer avec JavaScript
- Performance considerations

Exercice : Créer un module WebAssembly en Rust. L'utiliser dans une page web. Comparer performance avec JavaScript équivalent.

Jour 94 : Embedded Rust

Objectif : Rust pour systèmes embarqués

- #![no_std] programming
- embedded-hal
- Target embedded
- Interrupts et peripherals
- Memory-constrained programming

Exercice : Écrire un programme no_std simple. Simuler un environnement embarqué. Comprendre les contraintes.

Jour 95 : Procedural macros avancées

Objectif : Macros sophistiquées

- syn parsing avancé
- quote generation
- Macro hygiene
- Debugging macros
- Macro libraries

Exercice : Créer une macro procédurale complexe : builder pattern generator ou serialization framework custom.

Jour 96 : Type-level programming

Objectif : Types avancés

- Phantom types
- Type-state pattern
- Const generics
- GATs (Generic Associated Types)
- Type-level computations

Exercice : Implémenter un builder avec type-state pattern : états compilés garantissent utilisation correcte.

Jour 97 : Async advanced patterns

Objectif : Patterns async complexes

- Select et join macros
- Cancellation patterns
- Timeouts avancés
- Actor pattern
- Async drop (challenges)

Exercice : Implémenter un actor system simple. Messages entre actors, supervision, error handling.

Jour 98 : Planification projet final

Objectif : Design et architecture

- Choisir un projet ambitieux
- Design document complet
- Architecture et modules
- Choix technologiques
- Milestones et planning

Exercice : Documenter complètement le projet final : requirements, architecture diagrams, API specs, database schema, timeline.

PROJET FINAL (Jours 99-105) : 7 jours

Options de projets finaux

Option 1 : Système de cache distribué (Redis-like)

Architecture :

- Server TCP async avec Tokio
- Protocol personnalisé (RESP-like)
- Commands : GET, SET, DELETE, EXPIRE, KEYS
- Data structures : String, List, Set, Hash
- Persistence : AOF et snapshots
- Replication master-slave
- Pub/sub system

Features avancées :

- Clustering avec sharding
- Transactions (MULTI/EXEC)
- Lua scripting (rhai crate)
- Monitoring et metrics
- TLS support
- Benchmarks vs Redis

Qualité :

- Tests unitaires et intégration

- Property testing pour data structures
- Concurrent stress tests
- Documentation protocole
- Client library

Option 2 : Moteur de recherche full-text

Architecture :

- Inverted index avec skip lists
- Tokenization et stemming
- TF-IDF scoring
- Boolean queries
- Phrase search
- Fuzzy matching
- API REST async

Features avancées :

- Incremental indexing
- Distributed indexing
- Faceting et filtering
- Suggester pour autocompletion
- Highlighting
- Multi-language support
- Real-time updates

Performance :

- SIMD pour scoring

- Memory-mapped files
- Compression
- Benchmarks vs Elasticsearch

Option 3 : Container runtime (Docker-like)

Architecture :

- CLI avec Clap
- Namespaces Linux
- Cgroups pour resource limits
- Union filesystem (OverlayFS)
- Image management
- Container lifecycle
- Networking

Features :

- Build images depuis Dockerfile
- Registry pull/push
- Volume management
- Port mapping
- Container logs
- Stats et monitoring

Sécurité :

- Capabilities dropping
- Seccomp profiles
- AppArmor/SELinux

- User namespaces

Option 4 : Compilateur et VM

Architecture :

- Langage simple (Lisp-like ou autre)
- Lexer et parser
- AST et semantic analysis
- Bytecode compiler
- Stack-based VM
- Garbage collector

Features :

- Type system
- Standard library
- REPL interactif
- Debugger
- Optimizations
- JIT compilation (bonus)

Qualité :

- Parser tests exhaustifs
- VM tests
- Benchmark suite
- Example programs
- Language documentation

Option 5 : Database engine

Architecture :

- B+ tree pour index
- Page-based storage
- Buffer pool manager
- Transaction support (ACID)
- Query parser et executor
- SQL subset

Features :

- Multiple indexes
- Joins (nested loop, hash)
- Aggregations
- Query optimizer
- Concurrent transactions
- Recovery (WAL)

Performance :

- Lock-free structures
- MVCC pour isolation
- Benchmarks TPC-like

Exigences communes projet final

Code Quality :

- Architecture claire et modulaire
- Minimum 80% code coverage
- Clippy warnings = 0

- Documentation exhaustive
- README professionnel

Performance :

- Profiling et optimisation
- Benchmarks vs alternatives
- Flamegraphs inclus
- Memory profiling

Testing :

- Unit tests pour chaque module
- Integration tests
- Property-based testing
- Fuzzing pour parsers
- Stress tests

DevOps :

- CI/CD avec GitHub Actions
- Docker image optimisée
- Release binaries multi-platform
- Versioning sémantique

Documentation :

- Architecture document
- API documentation complète
- User guide
- Developer guide

- Examples et tutorials

Sécurité :

- Security audit
- Input validation partout
- No unsafe sans justification
- Dependency audit avec cargo-audit

Observabilité :

- Structured logging
- Metrics (prometheus)
- Tracing distribué (si applicable)
- Health checks

Semaine Bonus : Écosystème et spécialisations

Jour 106 : CLI tools avec Clap

- Clap derive API
- Subcommands
- Argument validation
- Shell completions
- Configuration files

Jour 107 : Game development

- Bevy engine
- ECS architecture
- Systems et components

- Sprites et physics
- Audio et input

Jour 108 : GUI programming

- egui pour immediate mode
- iced pour reactive
- tauri pour desktop apps
- Event handling
- State management

Jour 109 : Network protocols

- Implémenter HTTP/1.1
- Parser de protocol
- State machines
- Error handling
- Streaming responses

Jour 110 : Blockchain basics

- Hash chains
- Proof of work
- Transactions
- Wallet et signatures
- Consensus algorithms

Jour 111 : Machine learning

- ndarray pour arrays
- linfa pour ML

- Loading trained models
- Inference optimization
- ONNX runtime

Jour 112 : Audio processing

- cpal pour I/O audio
- DSP algorithms
- Synthesizers
- Effects processing
- Real-time constraints

Jour 113 : Video processing

- ffmpeg bindings
- Frame processing
- Encoding/decoding
- Filters
- Performance optimization

Jour 114 : Operating Systems

- Kernel development basics
- Bootloader
- Memory management
- Scheduling
- Drivers

Ressources complémentaires

Livres essentiels

1. **The Rust Programming Language** (The Book)
2. **Rust for Rustaceans** (Jon Gjengset)
3. **Programming Rust** (O'Reilly)
4. **Rust in Action** (Tim McNamara)
5. **Zero To Production In Rust** (Luca Palmieri)
6. **Hands-On Concurrency with Rust**

Documentation et références

- **doc.rust-lang.org** : documentation officielle
- **rust-lang.github.io/async-book/** : Async programming
- **rust-lang.github.io/nomicon/** : Rustonomicon (unsafe)
- **rust-lang.github.io/api-guidelines/** : API design
- **docs.rs** : documentation de toutes les crates

Sites d'apprentissage

- **rustlings** : exercices interactifs
- **exercism.org/tracks/rust** : exercices guidés
- **rust-lang.org/learn** : ressources officielles
- **fasterthanli.me** : articles approfondis
- **blog.rust-lang.org** : actualités

Crates essentielles à connaître

Async & Web :

- tokio, async-std

- axum, actix-web, rocket
- reqwest, hyper
- tower (middleware)
- tonic (gRPC)

Database :

- sqlx, diesel
- serde (serialization)
- mongodb, redis crates

CLI & TUI :

- clap, structopt
- indicatif (progress bars)
- console, dialoguer
- tui-rs, ratatui

Testing & Quality :

- criterion (benchmarks)
- proptest, quickcheck
- mockall (mocking)
- insta (snapshot testing)

Utilities :

- anyhow, thiserror (errors)
- tracing (logging)
- rayon (parallelism)
- crossbeam (concurrency)

Communauté Rust

- **users.rust-lang.org** : forum officiel
- **r/rust** : subreddit
- **Discord Rust Community**
- **This Week in Rust** : newsletter
- **RustConf** et meetups locaux

Guide de progression

Semaines 1-4 : Fondamentaux (28 jours)

Objectif : Maîtriser les bases uniques à Rust

- Ownership, borrowing, lifetimes
- Structs, enums, pattern matching
- Collections et error handling
- Traits et génériques

Validation :

- Tous les exercices compilent
- Tests passent
- Comprendre les messages du compilateur
- Projets hebdomadaires fonctionnels

Semaines 5-8 : Intermédiaire (28 jours)

Objectif : Organisation et types avancés

- Tests et documentation
- Modules et architecture

- Closures et iterators
- Smart pointers

Validation :

- Code bien organisé
- Tests automatisés
- Documentation claire
- Comfortable avec borrow checker

Semaines 9-11 : Avancé (21 jours)

Objectif : Concurrence et applications réelles

- Threads et async/await
- Web development
- Databases
- Sécurité

Validation :

- Applications complètes fonctionnelles
- Async code naturel
- Gestion d'erreurs robuste
- Performance acceptable

Semaines 12-14 : Expert (21 jours)

Objectif : Optimisation et maîtrise

- Performance tuning
- Sécurité cryptographique
- Sujets spécialisés

- Projet final ambitieux

Validation :

- Code production-ready
- Benchmarks et optimisations
- Architecture solide
- Portfolio impressionnant

Conseils pour réussir avec Rust

Combat le borrow checker

1. **Ne pas se décourager** : Il est strict mais vous protège
2. **Lire les messages** : Ils sont très informatifs
3. **Commencer simple** : Ownership avant lifetimes complexes
4. **Dessiner** : Visualiser ownership aide beaucoup
5. **Pratiquer** : Ça devient naturel avec le temps

Mindset Rust

1. **Explicit over implicit** : Rust préfère la clarté
2. **Zero-cost abstractions** : Performance sans compromis
3. **Fearless concurrency** : Le compilateur garantit la sûreté
4. **If it compiles, it usually works** : Forte confiance
5. **Error handling is mandatory** : Pas d'exceptions ignorées

Debugging

1. **cargo check** avant cargo build (plus rapide)
2. **cargo clippy** pour suggestions

3. **rust-analyzer** dans votre IDE
4. **println!** debugging est OK
5. **Compiler errors sont vos amis**

Performance

1. **Ne pas optimiser prématurément**
2. **Mesurer avec benchmarks**
3. **Profile avant d'optimiser**
4. **Release mode pour benchmarks**
5. **Iterator chains sont souvent optimisés**

Apprentissage continu

1. **Lire du code Rust** : GitHub, docs.rs
2. **Contribuer à l'open source**
3. **Suivre This Week in Rust**
4. **Participer aux discussions**
5. **Enseigner aux autres**

Évaluation et certification

Auto-évaluation par phase

Phase Débutant (Semaines 1-4) :

- [ ] Comprends ownership et borrowing
- [ ] Écris des fonctions sans warnings
- [ ] Utilise structs et enums naturellement
- [ ] Gère les erreurs avec Result
- [ ] Lis la documentation standard library

Phase Intermédiaire (Semaines 5-8) :

- [ ] Organise code en modules
- [ ] Écris des tests automatiquement
- [ ] Utilise traits pour abstraction
- [ ] Comprends les lifetimes
- [ ] Code avec iterators et closures

Phase Avancé (Semaines 9-11) :

- [ ] Écris code async confortablement
- [ ] Construis des web services
- [ ] Gère la concurrence avec confiance
- [ ] Intègre des databases
- [ ] Applique sécurité et cryptographie

Phase Expert (Semaines 12-14) :

- [ ] Profile et optimise code
- [ ] Comprends unsafe et quand l'utiliser
- [ ] Conçois architectures complexes
- [ ] Contribue à l'écosystème
- [ ] Mentore d'autres développeurs

Portfolio final

Minimum requis :

- 14 projets hebdomadaires sur GitHub
- 1 projet final documenté et déployé
- Tests et CI/CD configurés
- README professionnels
- Au moins 3 crates publiées (can be small)

Compétences validées

Après ces 14 semaines, vous serez capable de :

- ✅ Développer des applications Rust complexes
- ✅ Contribuer à des projets open source
- ✅ Écrire du code concurrent sûr
- ✅ Optimiser pour la performance
- ✅ Débugger efficacement
- ✅ Architecturer des systèmes
- ✅ Mentorer d'autres développeurs

Conclusion

Ce programme de **14 semaines** (98 jours d'exercices + 7 jours projet final) vous transformera en développeur Rust compétent. Avec **105 exercices quotidiens** et **15 projets**, vous aurez une maîtrise solide du langage.

Rust est difficile au début, mais incroyablement gratifiant. Le compilateur strict devient votre allié, et la confiance dans votre code est incomparable.

Prochaines étapes après le programme

1. **Spécialiser** : Choisir un domaine (systems, web, embedded)
2. **Contribuer** : Open source pour apprendre des meilleurs
3. **Construire** : Projets personnels ambitieux
4. **Partager** : Blogs, talks, mentoring
5. **Approfondir** : Unsafe, FFI, optimisations avancées

Total : 105+ exercices, 15 projets, la voie vers l'expertise Rust 🦀

Keep Rusting! La communauté Rust est accueillante et prête à aider.