Rust : maîtriser les fondamentaux

Développer des applications modernes, sûres et performantes

Stage pratique
Qualité du logiciel - Software Craftsmanship

Code :
RUST1

Durée :
3 jour(s) (21,00 heures)

Exposés : **40 %**
Cas pratiques : **40 %**
Echanges d'expérience : **20 %**

Inter-entreprises :
Prochaines sessions disponibles sur notre site web.
Tarif : 2 150,00 € HT / participant

Intra-entreprise :
Tarifs et dates sur demande.

DESCRIPTION

Rust s'impose aujourd'hui comme un langage de référence pour développer des logiciels à la fois fiables et performants, sans compromis sur la productivité. Adopté par des acteurs majeurs tels que Microsoft, Google, le noyau Linux ou encore Mozilla, il répond aux limites historiques du C et du C++, notamment en matière de sécurité mémoire, tout en offrant des performances comparables.

Cette formation de trois jours vous guide pas à pas, de l'installation à la prise en main des concepts clés : ownership, borrow checking, traits et génériques. Vous apprendrez à écrire un code sûr dès la compilation, capable d'éliminer en amont toute une classe de bugs critiques.

Que vous veniez du développement web, du logiciel système ou de l'embarqué, Rust s'adapte à votre contexte. À l'issue de la formation, vous serez en mesure de concevoir des outils en ligne de commande, des backends robustes et des bibliothèques réutilisables. Vous disposerez également de bases solides pour approfondir des sujets avancés comme l'asynchrone, le parallélisme ou le développement en environnement contraint (no_std)

OBJECTIFS PEDAGOGIQUES

- Concevoir et développer des logiciels fondamentaux (backends, outils CLI) fiables grâce à la maîtrise du système de propriété et d'emprunt de Rust et aux garanties de sécurité mémoire apportées par le compilateur
- Structurer et implémenter un code performant et réutilisable en mobilisant les traits et les génériques pour créer des abstractions zero-cost, et en adaptant ces solutions à différents contextes d'exécution (cloud, edge, embedded)
- Exploiter les matériels modernes (multi-coeurs, async I/O) tout en garantissant l'absence de problèmes d'accès concurrents

PUBLIC CIBLE

Toute personne souhaitant programmer en Rust.

PRE-REQUIS

Une expérience en programmation est requise.

Une familiarité avec les concepts de programmation système (gestion de

la mémoire, types, compilation) dans n'importe quel langage constitue un atout.

METHODE PEDAGOGIQUE
Formation avec apports théoriques, échanges sur les contextes des participants et retours d'expérience pratique des formateurs, complétés de travaux pratiques et de mises en situation.

PROFIL DES INTERVENANTS

Cette formation est dispensée par un·e ou plusieurs consultant·es d'OCTO Technology ou de son réseau de partenaires, expert·es reconnus des sujets traités.

Le processus de sélection de nos formateurs et formatrices est exigeant et repose sur une évaluation rigoureuse leurs capacités techniques, de leur expérience professionnelle et de leurs compétences pédagogiques.

MODALITÉS D'ÉVALUATION ET FORMALISATION À L'ISSUE DE LA FORMATION
L'évaluation des acquis se fait tout au long de la session au travers des ateliers et des mises en pratique. Afin de valider les compétences acquises lors de la formation, un formulaire d'auto-positionnement est envoyé en amont et en aval de celle-ci. Une évaluation à chaud est également effectuée en fin de session pour mesurer la satisfaction des stagiaires et un certificat de réalisation leur est adressé individuellement.

PROGRAMME PEDAGOGIQUE DETAILLE

Jour 1

DÉCOUVERTE DE RUST

- Présenter les grandes étapes de l'évolution de Rust à travers un aperçu historique synthétique
- Expliquer les motivations ayant conduit à l'émergence de Rust et sa place dans l'écosystème actuel
- Identifier et décrire les principaux atouts du langage : sécurité mémoire, performances et productivité
- Analyser les limites et compromis de Rust afin d'en comprendre les cas d'usage pertinents

INSTALLATION ET PRISE EN MAIN

- Installer et configurer l'environnement de développement Rust

à l'aide de Rustup, Cargo et des outils associés
- Mettre en place un environnement de travail ფუნქctionnel (IDE, extensions, configuration) adapté au développement Rust
- Créer, compiler et exécuter un premier programme Rust simple
- Mise en pratique :
  - Installer Rust et Cargo via Rustup
  - Configurer un environnement de développement avec VSCode et les extensions nécessaires
  - Écrire, compiler et exécuter un programme "Hello, World!" afin de valider le bon fonctionnement de l'installation

PLONGÉE DANS L'ÉCOSYSTÈME

- Expliquer et utiliser Cargo comme outil central pour la gestion de projet (build, exécution, dépendances)
- Compiler et exécuter efficacement une application Rust en maîtrisant les commandes essentielles
- Mettre en œuvre des pratiques de qualité en écrivant des tests, en documentant le code et en analysant les warnings avec les outils adaptés
- Appliquer les conventions de formatage automatique pour améliorer la lisibilité et la maintenabilité du code
- Gérer et optimiser les dépendances d'un projet Rust
- Mise en pratique à travers deux mini-applications : "Implémentation de tests unitaires, correction des warnings avec Clippy et documentation du projet"

TYPAGE DE DONNÉES

- Décrire et utiliser le système de types de Rust pour garantir la sûreté et la robustesse du code
- Manipuler et comparer différentes structures de données (tuples, slices, vecteurs, pointeurs) afin de choisir les plus adaptées à chaque usage
- Traiter et manipuler des chaînes de caractères en tenant compte des spécificités d'Unicode et des types propres à Rust
- Appliquer des conversions de types et définir des alias pour améliorer la lisibilité, la maintenabilité et la sécurité du code

EXPLOITATION DE LA SYNTAXE

- Utiliser les fondamentaux du langage (variables, structures de contrôle, fonctions) pour écrire des programmes Rust simples et structurés
- Comprendre la propriété et l'emprunt : références partagées et exclusives
- Structures complexes : Structs, enums et leurs utilisations

- Rust idiomatique : Pattern matching, implémentations et fermetures
- Mise en pratique : “Développer une application de conversion de température, en employant une variété de structures et syntaxes idiomatiques, tout en respectant les bonnes pratiques de tests unitaires, de formatage et de linting”

Jour 2

MODULARITÉ EN RUST

- Écrire un programme en une seule ligne, c’est possible !
- Quatre principes essentiels : lisibilité, maintenabilité, réutilisabilité et interopérabilité
- Spécificités et mécanismes d'import et de modularisation
- Créer des modules : définir, déclarer et intégrer des modules dans une application
- Visibilité et portée : contrôle d'accès et interactions entre modules
- Mise en pratique : “Réorganiser des applications existantes en modules et sous-modules clairement structurés. Appliquer les règles de visibilité, gérer les imports efficacement et concevoir un prélude pour simplifier l’usage des composants”

GESTION DES ERREURS

- La philosophie Rust : une véritable gestion des erreurs
- Distinguer et gérer les erreurs récupérables et irrécupérables
- Le trait Error en profondeur : centraliser la gestion d'erreurs
- Techniques avancées : propagation, erreurs personnalisées et bibliothèques spécialisées
- Mise en pratique : “Concevoir une stratégie idiomatique et complète de gestion des erreurs. Implémenter cette stratégie dans des applications existantes en anticipant différents scénarios, et choisir les bibliothèques les plus adaptées (ThisError, Anyhow, Eyre…)”

TRAITS ET GÉNÉRIQUES

- Découvrir les traits, sous-traits et leur rôle central
- Fonctions génériques : réutiliser du code à l'infini
- Mettez vos fruits dans le même panier avec les trait objects
- Sous le capot : monomorphisation et dispatch dynamique
- Utilisation avancée des traits : types associés et génériques
- Mise en pratique : “Améliorer des applications existantes en implémentant des traits standards tels que Display, Debug,

Default, Eq, Ord, From et Into”

Jour 3

COLLECTIONS DE DONNÉES EN RUST

- Maîtriser les principales collections (Vvecteurs, hashmaps, sets, etc.) et leurs cas d’usage
- Accéder aux données avec élégance et efficacité grâce aux Entries
- Techniques avancées : itérateurs et opérations courantes
- Construire des collections à partir d’un itérateur
- Mise en pratique : “Exploiter les traits Iterator et IntoIterator en réimplémentant fizzbuzz et fibonacci sous forme d’itérateurs, selon deux approches différentes”

MACROS

- (Re)Découvrez les macros natives et leur rôle dans l’écosystème Rust
- Les macros en Rust : sécurité et hygiène
- Macros déclaratives ou procédurales
- Écrire vos propres macros avec macro rules pour générer du code réutilisable

POUR ALLER PLUS LOIN

- Domaines avancés : unsafe, parallélisme, async, interopérabilité et développement embarqué
- Tour d’horizon des bibliothèques populaires et des ressources pour apprendre continuellement
- Mise en pratique : “Sérialiser et désérialiser les structures précédemment implémentées avec Serde dans différents formats (JSON, YAML, TOML…). Explorer les interactions avec d’autres langages et plateformes”

SÉANCE DE PRATIQUE

- Mettre en application les concepts abordés à travers une série d’exercices progressifs, allant de puzzles à difficulté croissante jusqu’à l’exploration complète de cartes

Accessibilité

L'inclusion est sujet important pour OCTO Academy.
Nos référent·es sont à votre disposition pour faciliter l'adaptation de votre formation à vos besoins spécifiques.
Pour les contacter : academy.accessibilite@octo.com